Statistics and Computing (2021) 31:4
https://doi.org/10.1007/s11222-020-09980-4

Scalable Bayesian inference for self-excitatory stochastic processes applied to big American gunfire data

Andrew J. Holbrook[1] · Charles E. Loeffler[2] · Seth R. Flaxman[3] · Marc A. Suchard[1,4,5]

Received: 13 May 2020 / Accepted: 2 December 2020

Abstract
The Hawkes process and its extensions effectively model self-excitatory phenomena including earthquakes, viral pandemics, financial transactions, neural spike trains and the spread of memes through social networks. The usefulness of these stochastic process models within a host of economic sectors and scientific disciplines is undercut by the processes' computational burden: complexity of likelihood evaluations grows quadratically in the number of observations for both the temporal and spatiotemporal Hawkes processes. We show that, with care, one may parallelize these calculations using both central and graphics processing unit implementations to achieve over 100-fold speedups over single-core processing. Using a simple adaptive Metropolis–Hastings scheme, we apply our high-performance computing framework to a Bayesian analysis of big gunshot data generated in Washington D.C. between the years of 2006 and 2019, thereby extending a past analysis of the same data from under 10,000 to over 85,000 observations. To encourage widespread use, we provide HPHAWKES, an open-source R package, and discuss high-level implementation and program design for leveraging aspects of computational hardware that become necessary in a big data setting.

Keywords Massive parallelization · GPU · SIMD · Spatiotemporal Hawkes process

1 Introduction

The gun violence epidemic in the USA is associated with over 30,000 deaths each year and over 650,000 deaths in the past twenty (Centers for Disease Control and Prevention 2020). Although a serious problem, mass shootings only account for a small fraction of these deaths, while gun-related homicides are most common in poor metropolitan areas (Bjerregaard and Lizotte 1995; National Research Council 2013). In 2005, for example, the highest per capita gun homicide rate in the country was 35.4 per 100,000 inhabitants in Washington D.C. (Federal Bureau of Investigation 2005). Despite its massive scale, the nature of gun-related violence and its impact on US public health remains poorly understood due, in part, to a paucity in the number of researchers focused on the field. In 2013, there were only 20 academic researchers in the USA focusing on gun violence "and most of them [were] economists, criminologists or sociologists" (Wadman 2013). This dearth in public health experts studying gun violence is largely due to the 1996 Dickey Amendment prohibiting the Centers for Disease Control and Prevention (CDC) from promoting gun control (Wadman 2013; Rubin 2016). Similarly, for researchers interested in studying gun violence, data availability has been a persistent issue (National Research Council 2005, 2013). While jurisdictions routinely report incidents where individuals are killed using firearms, non-fatal and near miss incidents, which vastly outweigh fatal firearm incidents, have been much less reliably reported.

But there are two reasons for (tempered) hope that we will better understand gun violence in the future. First, the federal budget for the 2020 fiscal year includes expenditures up to $25 million dollars to be split between the CDC and the National Institutes of Health for research in the reduction of

✉ Andrew J. Holbrook
aholbroo@g.ucla.edu

1 Department of Biostatistics, University of California, Los Angeles, Los Angeles, USA

2 Department of Criminology, University of Pennsylvania, Philadelphia, USA

3 Department of Mathematics, Imperial College London, London, UK

4 Department of Biomathematics, University of California, Los Angeles, Los Angeles, USA

5 Department of Human Genetics, University of California, Los Angeles, Los Angeles, USA

Published online: 12 January 2021

gun-related deaths and injuries, marking the first such expenditure since 1996 (Grisales 2019). Second, new kinds of data that may shed light on the nature of gun violence have become publicly available within the past decade. Examples of this recent expansion in gun violence data availability include local police department open data portals, crowdsourced gun violence reporting systems and journalistic data initiatives. One new source of data, acoustic gunshot locator systems (AGLS; Showen (1997)) uses a spatially distributed network of acoustic sensors to triangulate the locations of gunshots in space and time, thus overcoming the fact that the majority of gunshots go unreported to law enforcement (Mares and Blackburn 2012). And so, a new challenge arises: combining the massive scale of American gun violence with the fidelity of AGLS results in a potential deluge of big American gunfire data, and we must develop the computational and statistical techniques to effectively analyze them.

Analysis of gun violence in the USA has long relied on a range of modeling approaches drawn from spatiotemporal statistics including classical Knox tests, K-functions and more recent developments such as Gaussian processes (Ratcliffe and Rengert 2008; Flaxman 2015). Using these tests, scholars have sought to detect and study the degree of stability of gun violence clusters, sometimes referred to as gun violence hotspots. They have also explored whether gun violence diffuses in space and time as well as within social networks of susceptible places and populations and whether public health and law enforcement interventions designed to reduce the toll of gun violence are effective in diminishing its incidence. Examples of such interventions include violence interruption programs, focused deterrence initiatives as well as more traditional policing interventions. All of these rely on spatiotemporal measures to generate evidence of both theoretical and policy significance. Using these reliable methods, as they have been successfully used in other public health domains, scholars have learned that only some types of gun violence reliably cluster and that at least some violence can be disrupted (Park et al. 2019).

At the same time, these implementations draw heavily on the availability of relatively sparse point process data or sensibly aggregated point process data to enable both inferential and predictive work. However, with the arrival of newer and higher resolution data sources such as AGLS, many oft-posed research questions need to be revisited in order to test whether the assumptions built into classical analyses hold up. Furthermore, some research questions that have been left unanswered due to the challenge of answering them using data measured with relatively low spatial and temporal resolution—to say nothing of missing data due to non-reporting—can now be explored. Key unresolved questions include the exact scales at which violence diffuses. Despite decades of research on the spatiotemporal patterns of gun violence, it remains an incompletely understood phenomenon. Some models report high levels of contagious diffusion of gun violence. Others report much lower levels. Similarly, policymakers remain split on which of these models most accurately describes the realities of gun violence in their cities. Relying on studies showing the diffusion of gun violence, policymakers have implemented violence interruption programs designed to halt the diffusion of gun violence. By contrast, policymakers relying on studies showing lower levels of diffusion have emphasized the need to address underlying community risk factors. Improving spatiotemporal models of gun violence, including gunshots, will support refinements of both theoretical models and related policy implementations.

As a case study in the spatiotemporal analysis of big gunfire data, we consider the Washington D.C. ShotSpotter AGLS dataset (Petho et al. 2013) consisting of over 85,000 potential gunfire events from 2006 to 2019. A previous analysis of these data (Loeffler and Flaxman 2018) restricts itself to a relatively small subset of around 9000 events occurring in the years 2010 through 2012. That same paper seeks to determine whether evidence exists for gun violence being contagious in the sense of bursts of diffusions through the urban landscape. We follow Loeffler and Flaxman (2018) and model this contagiousness using the *self-excitatory* spatiotemporal Hawkes process (Reinhart 2018), the computational complexity of which, unfortunately, scales quadratically in the number of observed events. As a result, scaling model calculations to all 85,000 events is difficult, but we overcome this challenge with the aid of massive parallelization and cutting-edge computational hardware.

The temporal Hawkes process (Hawkes 1971b, a, 1972) and its extensions are stochastic point processes that effectively model phenomena that are *self-excitatory* in nature. Given an earthquake, we expect to observe aftershocks soon after and close to the epicenter and a meme that is 'going viral' triggers a cascade of 'likes' that traverses the edges connecting a social network. Similarly, a diffusion of biological viruses across a human landscape also exhibits self-excitatory behavior, where an infected student or coworker often results in infected students or coworkers. Hawkes processes and extensions have successfully modeled earthquakes (Hawkes 1973; Ogata 1988; Zhuang et al. 2004), viral memes (Yang and Zha 2013; Mei and Eisner 2017), neural activity (Linderman and Adams 2014; Truccolo 2016; Linderman et al. 2017), viral epidemics (Kim 2011; Meyer and Held 2014; Choi et al. 2015; Rizoiu et al. 2018; Kelly et al. 2019) and financial transactions (Embrechts et al. 2011; Chavez-Demoulin and McGill 2012; Hardiman et al. 2013; Hawkes 2018).

Due to the wide, multi-sector use of the entire family of extended Hawkes process models, we believe that a demonstration of their natural parallelizability will be beneficial to theoreticians and practitioners alike. Specifically,

we use Bayesian inference (Rasmussen 2013; Linderman and Adams 2014) to learn posterior distributions of our spatiotemporal Hawkes process model parameters conditioned on tens of thousands of observed events. Our simple Markov chain Monte Carlo (MCMC; Metropolis et al. (1953); Hastings (1970)) algorithm requires repeated likelihood evaluations, each of which scales quadratically in computational complexity. Overcoming this bottleneck in a big data setting is the chief contribution of our work.

A robust literature exists for parallel implementations in statistical computing: Suchard and Rambaut (2009), Suchard et al. (2010a) and Suchard et al. (2010b) perform optimization and Bayesian inference using graphics processing units (GPUs); Lee et al. (2010) and Zhou et al. (2010) use the same hardware for sequential Monte Carlo and statistical optimization, respectively; and Beam et al. (2016) apply GPUs to the evaluation of the multi-nomial likelihood and its gradient. More recently, Warne et al. (2019) explore the use of central processing unit (CPU)-based single instruction, multiple data (SIMD) vectorization in various tasks within Bayesian inference, and Holbrook et al. (2019) use GPUs, multi-core CPUs and SIMD vectorization to accelerate MCMC for Bayesian multi-dimensional scaling with millions of data points. In a similar manner, we develop a high-performance computing framework for scalable MCMC for the spatiotemporal Hawkes process using many-core GPU, multi-core CPU and SIMD vectorization based implementations. To increase the impact of our work, we provide this high-performance computing framework as HPHAWKES, a rudimentary, open-source R package freely available at https://github.com/suchard-group/hawkes.

We note that White and Porter (2014) also consider GPU parallel implementations of Bayesian inference for a self-excitatory model and that our current work differs substantially from the content of that paper. First and from a computational standpoint, we present three different and practical parallelization approaches (multi-core and vectorized CPU and many-core GPU computing). While White and Porter (2014) compare GPU performance to an interpreted R language implementation, potentially overestimating speedups with respect to a compiled language, single-core CPU baseline by as much as a factor of ten, we compare GPU performance to non-vectorized and vectorized single-core and multi-core C++ implementations to paint a richer picture of relative hardware capabilities. Further, our model is spatiotemporal, rather than purely temporal, and does not rely on temporal binning. We also demonstrate the application of our high-performance computing framework to 85,000+ observations, compared to the roughly 5000 observations of White and Porter (2014). Not only do we develop tools for Bayesian inference, we also develop parallel computing methods for post-processing of MCMC samples to obtain interpretable results for individual events (Algorithms 4 and 5). Finally, we fully detail the inner workings of our parallelization strategies (Algorithms 2–5) with a view to helping the reader understand the nature of parallel computing and *why* it is appropriate for the broader class of self-excitatory point processes.

2 Methods

2.1 Model

The spatiotemporal Hawkes process describes the joint distribution of random variables $(\mathbf{x}, t) \in \mathbb{R}^D \times \mathbb{R}^+$ in space and time as an inhomogeneous Poisson process (Daley 2003; Daley and Vere-Jones 2007) with intensity function

$$\lambda(\mathbf{x}, t) = \mu(\mathbf{x}, t) + \sum_{t_n < t} g(\mathbf{x} - \mathbf{x}_n, t - t_n)$$

conditioned on observations $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)$. In this formulation, $\mu(\cdot, \cdot)$ is the background or endemic rate, and $g(\cdot, \cdot)$ is the triggering function describing the self-excitatory nature of the process. We follow Mohler (2014) and Loeffler and Flaxman (2018) in the use of a triggering function that is exponential in time and Gaussian in space when modeling crime data:

$$\lambda(\mathbf{x}, t) = \mu(\mathbf{x}, t) + \frac{\theta\omega}{h^D} \sum_{t_n < t} e^{-\omega(t - t_n)} \phi\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right).$$

Parameters ω, h and θ are strictly positive, and we call $1/\omega$ and h the temporal and spatial bandwidths belonging to the conditional rate function's self-excitatory term. We further opt for a flexible Gaussian kernel smoother to model the background rate

$$\mu(\mathbf{x}, t) = \frac{\mu_0}{\tau_x^D \tau_t} \sum_{n=1}^{N} \phi\left(\frac{\mathbf{x} - \mathbf{x}_n}{\tau_x}\right) \cdot \phi\left(\frac{t - t_n}{\tau_t}\right)$$

with τ_x and τ_t the spatial and temporal bandwidths corresponding to the endemic background rate. Taken together, μ_0 and θ describe the extent to which the process is self-excitatory in nature. Denoting $\Theta = (\mu_0, \tau_x, \tau_t, \theta, \omega, h)$, the likelihood (Daley 2003) for data $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)$ is

$$\begin{aligned}\mathcal{L}(\Theta) &= \exp\left(-\int_{\mathbb{R}^D} \int_0^{t_N} \lambda(\mathbf{x}, t)\, \mathrm{d}t\, \mathrm{d}\mathbf{x}\right) \prod_{n=1}^{N} \lambda(\mathbf{x}_n, t_n) \\ &:= e^{-\Lambda(t_N)} \cdot \prod_{n=1}^{N} \lambda_n.\end{aligned}$$

Here, we have chosen to integrate over the entirety of $\mathbb{R}^D$ rather than a relevant subset. This choice potentially leads to

biased inference and should be regarded as an approximation when measurement over $\mathbb{R}^D$ is incomplete (Schoenberg 2013). Our intensity function separates in space and time, so the integral $\Lambda(t_N)$ factorizes. The spatial integral is unity, and Laub et al. (2015) (Sect. 3.2) demonstrate the closed-form solution to the self-excitatory component's temporal integral with exponential triggering function:

$$\begin{aligned}\Lambda(t_N) &= \mu_0 \sum_{n=1}^{N} \left(\Phi\left(\frac{t_N - t_n}{\tau_t}\right) - \Phi\left(\frac{-t_n}{\tau_t}\right)\right) - \theta \sum_{n=1}^{N} \left(e^{-\omega\,(t_N - t_n)} - 1\right) \\ &= \sum_{n=1}^{N} \left(\mu_0 \left(\Phi\left(\frac{t_N - t_n}{\tau_t}\right) - \Phi\left(\frac{-t_n}{\tau_t}\right)\right) - \theta \left(e^{-\omega\,(t_N - t_n)} - 1\right)\right) \\ &:= \sum_{n=1}^{N} \Lambda_n\end{aligned}$$

Thus, we are able to calculate the log likelihood

$$\begin{aligned}\ell(\Theta) &= -\Lambda(t_N) + \sum_{n=1}^{N} \log \lambda_n \\ &= \sum_{n=1}^{N} \left\{ \log \left[\sum_{n'=1}^{N} \left(\frac{\mu_0}{\tau_x^D \tau_t} \phi\left(\frac{\mathbf{x}_n - \mathbf{x}_{n'}}{\tau_x}\right) \cdot \phi\left(\frac{t_n - t_{n'}}{\tau_t}\right) \right.\right.\right. \\ &\quad \left.\left.\left. + \frac{\theta\omega\, \mathcal{I}_{[t_{n'} < t_n]}}{h^D} e^{-\omega\,(t_n - t_{n'})} \phi\left(\frac{\mathbf{x}_n - \mathbf{x}_{n'}}{h}\right)\right)\right] - \Lambda_n \right\} \\ &:= \sum_{n=1}^{N} \left[\log \left(\sum_{n'=1}^{N} \lambda_{nn'} \right) - \Lambda_n \right] := \sum_{n=1}^{N} \ell_n, \end{aligned} \tag{1}$$

which we use for Bayesian inference in the context of a simple MCMC algorithm (Sect. 2.2). The likelihood's double summation over indices n and n' results in $\mathcal{O}(N^2)$ computational complexity: evaluation of the rate function for each fixed n is linear in complexity, and the outer sum over these same n is again linear. We overcome this computational burden by developing parallel implementations of likelihood calculations on cutting-edge computational hardware (Sect. 2.3). We also develop parallel implementations to compute the vector of probabilities π_n that each individual event generates from self-excitation rather than from the background process:

$$\pi_n = \frac{\lambda_n - \mu_n}{\lambda_n} := \frac{\xi_n}{\lambda_n}, \tag{2}$$

where ξ_n denotes the self-excitatory component of rate λ_n. For each n, π_n is a function of all $N-1$ other observations, so computing the entire vector is $\mathcal{O}(N^2)$. Moreover, each π_n is a function of Θ, and we take the posterior distribution of each π_n to be a key interpretable of our analysis. Given an MCMC sample $\Theta^{(1)}, \ldots, \Theta^{(S)}$, obtaining a posterior sample $\pi_n^{(s)}$ for $n = 1, \ldots, N$ and $s = 1, \ldots, S$ is $\mathcal{O}(N^2 S)$, again necessitating the cutting-edge computational hardware of Sect. 2.3.

To facilitate comparisons with Loeffler and Flaxman (2018), we follow their specification and equip μ_0 and θ with truncated normal priors with a lower bound of 0 and standard deviations of 1 and 10, respectively. We lend truncated normal priors to ω and $1/h$ again with a lower bound of 0 and with a standard deviation of 10 for both. Finally, we also follow that paper in setting the background rate's temporal and spatial lengthscales τ_t and τ_x to be 14 days and 1.6 km. While these settings accomplish our goal of an 'apples to apples' comparison with the results of Loeffler and Flaxman (2018), we note that such an approach may lead to biased inference for the parameters h, θ and ω as the model overcompensates for a misspecified background rate (Reinhart and Greenhouse 2018, Section 4.3). Importantly, the same likelihood calculations apply for inferring τ_x and τ_t, and our engineering and its resulting speedups therefore hold as well.

2.2 Inference

Algorithm 1 describes the simple, adaptive Metropolis–Hastings algorithm (Haario et al. 2001; Roberts and Rosenthal 2009) with random scan univariate proposals we use to generate posterior realizations for ω, h, θ and μ_0. Of the different algorithms described in the extensive adaptive MCMC literature, some of the simplest work by tuning the proposal distribution to obtain a target acceptance rate Roberts and Rosenthal (2009). Following (Gelman et al. 1996), we target an acceptance rate of 0.44 (Algorithm 1, Step 6d) for each of our four univariate proposals. We accomplish this while guaranteeing the *diminishing adaptation* criterion of Roberts and Rosenthal (2007) by increasing adaptation intervals at a super-linear rate (Algorithm 1, Step 6l). For any interesting posterior distribution conditioned on even moderately sized data, the algorithm's computational bottleneck is the calculation of the likelihood function in Step 5a. For most models belonging to the Hawkes process family, the computational complexity of this step is quadratic in the number of observations ($\mathcal{O}(N^2)$), and for our specific model this fact arises from the double summation of Eq. (1). In the following section, we discuss the multiple parallelization strategies we use to overcome this rate-limiting step.

2.3 Parallelization

To parallelize the Hawkes process likelihood of Eq. (1) and circumvent its $\mathcal{O}(N^2)$ computational complexity, we take a hardware oriented approach that uses four broad rules of thumb (Holbrook et al. 2019):

1. we design our code to assign calculations of stereotyped and ostensibly independent terms to independent cores; as such, we target the N^2 $\lambda_{nn'}$ terms of Eq. (1) for simultaneous processing insofar as the hardware supports;

Algorithm 1 A simple adaptive Metropolis–Hastings algorithm:

Produces a Markov chain of D-dimensional states $\Theta^{(s)} > \mathbf{0}$ for $s = 1, \ldots, S$ with target density $p(\cdot)$. Following initialization, each iteration uses a random scan to generate a univariate proposal, accepts or rejects that proposal using a Metropolis–Hastings accept–reject step and updates the D univariate proposal distributions with decreasing regularity. We use 0.44 as target acceptance rate following Gelman et al. 1996).

1: Initialize algorithmic quantities:
  a: Markov chain state counter $s \leftarrow 1$
  b: Markov chain state $\Theta^{(s)} = (\theta_1^{(s)}, \ldots, \theta_D^{(s)}) \leftarrow \Theta > \mathbf{0}$
  c: adaptation interval bounds $\mathbf{b} = (b_1, \ldots, b_D) \leftarrow (5, \ldots, 5)$
  d: adaptation interval counters $\mathbf{l} = (l_1, \ldots, l_D) \leftarrow (0, \ldots, 0)$
  e: acceptance counter $\mathbf{a} = (a_1, \ldots, a_D) \leftarrow (0, \ldots, 0)$
  f: truncated normal proposal standard deviations $\mathbf{v} = (v_1, \ldots, v_D) \leftarrow (1, \ldots, 1)$
2: **for** s in 1 : S **do**
3: Randomly select dth parameter to update: $d \sim$ Uniform$(1, \ldots, D)$
4: Generate proposal state $\Theta^* \sim q(\Theta^*|\Theta^{(s)})$:
  a: $\theta_d^* \sim \text{Normal}(\theta_d^{(s)}, v_d) \cdot \mathrm{I}(\theta_d^* > 0)$
  b: $\Theta^* \leftarrow (\theta_1^{(s)}, \ldots, \theta_d^*, \ldots, \theta_D^{(s)})$
5: Metropolis–Hastings accept–reject step:
  a: Calculate acceptance criterion: $r_1 \leftarrow \left(p(\Theta^*)\, q(\Theta^{(s)}|\Theta^*)\right) / \left(p(\Theta^{(s)})\, q(\Theta^*|\Theta^{(s)})\right)$
  b: Generate cut-off variable $u \sim$ Uniform$(0, 1)$
  c: **if** $u < r_1$ **then**
  d: $\Theta^{(s+1)} \leftarrow \Theta^*$
  e: $a_d \leftarrow a_d + 1$
  f: **else**
  g: $\Theta^{(s+1)} \leftarrow \Theta^{(s)}$
  h: **end if-else**
6: Update adaptation parameters:
  a: Increment adaptation interval counter $l_d \leftarrow l_d + 1$
  b: **if** $l_d = b_d$ **then**
  c: Calculate proportion of acceptances $r_2 \leftarrow a_d / b_d$
  d: Calculate adaptation ratio $r_3 \leftarrow r_2 / 0.44$
  e: **if** $r_3 > 2$ **then**
  f: $r_3 \leftarrow 2$
  g: **end if**
  h: **if** $r_3 < 0.5$ **then**
  i: $r_3 \leftarrow 0.5$
  j: **end if**
  k: Update proposal standard deviation $v_d \leftarrow r_3 \cdot v_d$
  l: Increase adaptation interval bound $b_d \leftarrow b_d^{1.1}$
  m: Reset adaptation interval counter $l_d \leftarrow 0$
  n: Reset acceptance counter $a_d \leftarrow 0$
  o: **end if**
7: **end for**

Algorithm 2 Parallel computation of Hawkes process likelihood:

Uses multiple central processing unit (CPU) cores along with loop vectorization to compute log likelihood. For double-precision floating point, the algorithm uses either SSE or AVX vectorization to make $j = 2$ or 4 long jumps and cut loop iterations by one-half or three-fourths, respectively. Here, B is the number of CPU threads available. Symbols ℓ, λ and Λ appear in Eq. (1).

1: **parfor** $b \in \{1, \ldots, B\}$ **do**
2: $\ell_b \leftarrow 0$
3: **if** $b \neq B$ **then**
4: $Upper \leftarrow b\lfloor N/B \rfloor$
5: **else**
6: $Upper \leftarrow \lceil N/B \rceil$
7: **end if**
8: **for** $n' \in \{(b-1)\lfloor N/B \rfloor + 1, \ldots, Upper\}$ **do**
9: copy $\mathbf{x}_{n'}, t_{n'}$ to cache
10: $\lambda_{n'} \leftarrow \mathbf{0}$ ▷ vector of length j
11: $n \leftarrow 1$
12: **while** $n < N$ **do**
13: $j \leftarrow \min(j, N - n)$
14: copy $\mathbf{x}_{n:(n+j)}, t_{n:n+j}$ to cache
15: $\Delta_{n'n} : \Delta_{n'n:(n+j-1)} \leftarrow (\mathbf{x}_{n'} - \mathbf{x}_n) : (\mathbf{x}_{n'} - \mathbf{x}_{n+j-1})$ ▷ vectorized subtraction
16: calculate $\delta_{n'n} : \delta_{n'(n+j-1)}$ ▷ vectorized multiplication, see Algorithm 3
17: calculate $\lambda_{n'n} : \lambda_{n'(n+j-1)}$ ▷ vectorized evaluation, see Algorithm 3
18: $\lambda_{n'} \leftarrow \lambda_{n'} + \lambda_{n'n} : \lambda_{n'(n+j-1)}$ ▷ vectorized addition
19: $n \leftarrow n + j$
20: **end while**
21: $\ell_b \leftarrow \ell_b + \log\left(\sum\{\lambda_{n'}\}\right) - \Lambda_{n'}$
22: **end for**
23: **end parfor**
24: $\ell(\Theta) \leftarrow \sum_b \ell_b$

2. we identify rate-limiting floating point calculations and perform them in parallel across vectors of inputs, thus providing an additional level of parallelization over and beyond the use of multiple cores; for our model, the rate-limiting floating point calculations occur in the evaluation of $\exp(\cdot)$ in the individual $\lambda_{nn'}$s;
3. when calculations require the use of individual data multiple times, we store these data so as to encourage fast reuse; for example, the calculation of λ_n requires the evaluation of N $\lambda_{nn'}$ terms, each of which depends on $\mathbf{x}_n$ and t_n;
4. we avoid costly storage of intermediate terms such as the individual $\lambda_{nn'}$ within our calculations and only store their running sum.

Different kinds of computational hardware capitalize on and facilitate these general strategies to different degrees. Cluster computing scales to 1000s of CPUs connected by Ethernet or Infiniband networks, each CPU having its own random access memory (RAM). The scale of such a cluster is undercut, however, by latency arising from communication between cluster nodes. If one divides a computing task into two parts, the first being parallelizable and having sequential cost c_0, the second being non-parallelizable and having cost c_1, then one can accelerate compute time by sharing c_0 between ν nodes. Unfortunately, Amdahl's law (Amdahl 1967) says that the resulting wall time c exhibits the bound

Algorithm 3 Parallel computation of Hawkes process likelihood:
Calculates the log likelihood with multiple levels of parallelization on graphics processing unit (GPU). In practice, we specify $B = 128$ to be the size of the GPU work groups. Symbols ℓ, λ and Λ appear in Eq. (1).

1: Calculate observation-specific contributions to likelihood ℓ_n:
  a: **parfor** $n \in \{1, \ldots, N\}$ **do**
  b: copy $\mathbf{x}_n, t_n$ to local ▷ B threads
  c: **parfor** $N' \in \{1, \ldots, \lfloor N/B \rfloor\}$ **do**
  d: $n' \leftarrow N'$
  e: $\lambda_{nN'} \leftarrow 0$
  f: **while** $n' < N$ **do**
  g: copy $\mathbf{x}_{n'}, t_{n'}$ to local ▷ B threads
  h: $\Delta_{nn'} \leftarrow \mathbf{x}_n - \mathbf{x}_{n'}$ ▷ vectorized subtraction
  i: calculate $\delta_{nn'} = \sqrt{\sum \Delta_{nn'} \circ \Delta_{nn'}}$ ▷ vectorized multiplication
  j: $\lambda_{nN'} \leftarrow \lambda_{nN'} + \lambda_{nn'}$ ▷ $\lambda_{nn'}$ a function of $\delta_{nn'}$, t_n and $t_{n'}$
  k: $n' \leftarrow n' + B$
  l: **end while**
  m: **end parfor**
  n: $\lambda_n \leftarrow \sum_{N'} \lambda_{nN'}$ ▷ binary tree reduction on chip
  o: $\ell_n \leftarrow \log \lambda_n - \Lambda_n$
  p: **end parfor**
2: Sum up all N observation-specific contributions ℓ_n:
  a: **parfor** $N' \in \{1, \ldots, \lfloor N/B \rfloor\}$ **do**
  b: $n' \leftarrow N'$
  c: $\ell_{N'} \leftarrow 0$
  d: **while** $n' < N$ **do**
  e: copy $\ell_{n'}$ to local ▷ B threads
  f: $\ell_{N'} \leftarrow \ell_{N'} + \ell_{n'}$
  g: $n' \leftarrow n' + B$
  h: **end while**
  i: **end parfor**
  j: $\ell(\Theta) \leftarrow \sum_{N'} \ell_{N'}$ ▷ binary tree reduction on chip

Algorithm 4 Parallel computation of self-excitatory probabilities:
Uses multiple central processing unit (CPU) cores along with loop vectorization to compute N self-excitatory probabilities π_n. For double-precision floating point, the algorithm uses either SSE or AVX vectorization to make $j = 2$ or 4 long jumps and cut loop iterations by one-half or three-fourths, respectively. Here, B is the number of CPU threads available. Symbols π_n, μ_n and ξ_n appear in Eq. (2).

1: **parfor** $b \in \{1, \ldots, B\}$ **do**
2: **if** $b \neq B$ **then**
3: $Upper \leftarrow b\lfloor N/B \rfloor$
4: **else**
5: $Upper \leftarrow \lceil N/B \rceil$
6: **end if**
7: **for** $n' \in \{(b-1)\lfloor N/B \rfloor + 1, \ldots, Upper\}$ **do**
8: copy $\mathbf{x}_{n'}, t_{n'}$ to cache
9: $\mu_{n'} \leftarrow \mathbf{0}$ ▷ vector of length j
10: $\xi_{n'} \leftarrow \mathbf{0}$ ▷ vector of length j
11: $n \leftarrow 1$
12: **while** $n < N$ **do**
13: $j \leftarrow \min(j, N - n)$
14: copy $\mathbf{x}_{n:(n+j)}, t_{n:n+j}$ to cache
15: $\Delta_{n'n} : \Delta_{n'n:(n+j-1)} \leftarrow (\mathbf{x}_{n'} - \mathbf{x}_n) : (\mathbf{x}_{n'} - \mathbf{x}_{n+j-1})$ ▷ vectorized subtraction
16: calculate $\delta_{n'n} : \delta_{n'(n+j-1)}$ ▷ vectorized multiplication, see Algorithm 3
17: calculate $\mu_{n'n} : \mu_{n'(n+j-1)}$
18: calculate $\xi_{n'n} : \xi_{n'(n+j-1)}$
19: $\mu_{n'} \leftarrow \mu_{n'} + \mu_{n'n} : \mu_{n'(n+j-1)}$ ▷ vectorized addition
20: $\xi_{n'} \leftarrow \xi_{n'} + \xi_{n'n} : \xi_{n'(n+j-1)}$ ▷ vectorized addition
21: $n \leftarrow n + j$
22: **end while**
23: $\pi_{n'} \leftarrow \xi_{n'}/(\mu_{n'} + \xi_{n'})$ ▷ vectorized addition, division and assignment
24: **end for**
25: **end parfor**

$$c \geq c_0/\nu + c_1$$

on account of latency arising from parallel tasks finishing at different times and additional communication between nodes. Indeed, for iterative algorithms such as MCMC, the lower bound on c becomes worse for every increasing iteration. Such inefficiencies often result in diminishing returns for large clusters, which can require significant financial investments nonetheless (Suchard et al. 2010a).

Given the latencies arising from iterative algorithms on large distributed-computing environments, we focus on the use of less expensive and more widely owned computing hardware to parallelize the evaluation of $\ell(\Theta)$, the bottleneck of our MCMC algorithm. First, we use the multiple cores and SIMD vectorization supported by most modern CPUs that are available in standard desktop computers. Second, we use the thousands of cores available in contemporary general-purpose GPUs to achieve massive parallelization. Specifically, we must use this hardware to parallelize the many *transformations* and *reductions* implied by Eq. (1). For a fixed index n, reading $\mathbf{x}_n$, t_n, $\mathbf{x}_{n'}$ and $t_{n'}$ from global memory and evaluating $\lambda_{nn'}$ is a transformation. Thus, we require N transformations to compute the N terms within the inner summation of Eq. (1). Following these transformations, a reduction maps from the individual $\lambda_{nn'}$s to their sum λ_n. A further transformation reads t_N, $\mathbf{x}_n$ and t_n from memory, computes Λ_n from them and adds $\log(\lambda_n)$ and Λ_n to obtain ℓ_n. A final reduction sums over all N ℓ_n to obtain the likelihood $\ell(\Theta)$. Regardless of the hardware type, we attack these transformation reductions with the same general principles: we perform rate-limiting floating point operations such as those involved in the evaluation of $\lambda_{nn'}$ in parallel; we keep data in fast access memory when we require reuse (notice how $\mathbf{x}_n$ and t_n appear in both transformations); and we use running summations to avoid costly reading and writing of intermediate values such as $\lambda_{nn'}$.

Algorithm 5 Parallel computation of self-excitatory probabilities:
Calculates N self-excitatory probabilities π_n from single parameter value Θ with multiple levels of parallelization on graphics processing unit (GPU). In practice, we specify $B = 128$ to be the size of the GPU work groups. Symbols π_n, μ_n and ξ_n appear in Eq. (2).

parfor $n \in \{1, \ldots, N\}$ **do**
  copy $\mathbf{x}_n, t_n$ to local ▷ B threads
  parfor $N' \in \{1, \ldots, \lfloor N/B \rfloor\}$ **do**
    $n' \leftarrow N'$
    $\mu_{nN'} \leftarrow 0$
    $\xi_{nN'} \leftarrow 0$
    while $n' < N$ **do**
      copy $\mathbf{x}_{n'}, t_{n'}$ to local ▷ B threads
      $\Delta_{nn'} \leftarrow \mathbf{x}_n - \mathbf{x}_{n'}$ ▷ vectorized subtraction
      calculate $\delta_{nn'} = \sqrt{\sum \Delta_{nn'} \circ \Delta_{nn'}}$ ▷ vectorized multiplication
      $\mu_{nN'} \leftarrow \mu_{nN'} + \mu_{nn'}$ ▷ $\mu_{nn'}$ a function of $\delta_{nn'}$, t_n and $t_{n'}$
      $\xi_{nN'} \leftarrow \xi_{nN'} + \xi_{nn'}$ ▷ $\xi_{nn'}$ a function of $\delta_{nn'}$, t_n and $t_{n'}$
      $n' \leftarrow n' + B$
    end while
  end parfor
  $\mu_n \leftarrow \sum_{N'} \mu_{nN'}$ ▷ binary tree reduction on chip
  $\xi_n \leftarrow \sum_{N'} \xi_{nN'}$ ▷ binary tree reduction on chip
  $\pi_n \leftarrow \xi_n/(\mu_n + \xi_n)$
end parfor

2.3.1 Multi-core CPUs

Contemporary desktops and servers have sockets for as many as 8 CPU chips. These CPU chips contain 1 to 72 independent processing units called cores, each of which can perform different operations in parallel, and each chip contains three (or more) levels of memory cache, L1, L2 and L3, that balance the rate of data transfer or *memory bandwidth* with the amount of data storage available. Typically, each core has its own L1 and L2 cache, where L1 has higher memory bandwidth but less storage than L2. Cores on the same chip usually share L3 cache, which has even less memory bandwidth and even more storage than L2. A memory bus connects on-chip cache to RAM, the bandwidth of which is significantly smaller than the total rate of numerical operations across cores. In a big data setting, memory bandwidth becomes a bottleneck for even the most numerically intensive tasks.

Many programming languages contain software libraries that enable the computational statistician to communicate with a computer's operating system and coordinate the behavior of multiple cores in the performance of independent tasks. We use THREADING BUILDING BLOCKS (TBB) (Reinders 2007), an open-source and cross-platform C++ library, for multi-core parallelization, and the R package RCPPPARALLEL makes TBB available to R developers (Allaire et al. 2016). These packages help to parallelize the transformation reductions of Eq. (1) by partitioning the task into T threads, for T less than or equal to the total number of cores of the multi-core environment. Each thread is limited in the rate at which it performs the rate-limiting floating point operations but has fast and unimpeded access to L1 and L2 caches. Specifically, we use TBB to assign calculation of elements $\lambda_{nn'}$, $n' = 1, \ldots, N$ to the same thread, so that a thread loads $\mathbf{x}_n$ and t_n to an on-chip register for reuse N times. The same thread obtains λ_n with a running summation of the $\lambda_{nn'}$ that avoids storage of intermediate values. After computing λ_n, the exact same thread computes a partial sum from a subset of the ℓ_ns and writes the partial sum to RAM. Finally, a single thread sums the T partial sums in a fast serial reduction. Algorithm 2 combines this multi-core implementation with within-core vectorization. Algorithm 4 is similar to Algorithm 2 and computes the vector of self-excitatory probabilities π_n.

2.3.2 Within-core vectorization

One can further accelerate multi-core CPU processing with the aid of vector or SIMD processing (Warne et al. 2019; Holbrook et al. 2019), in which the CPU simultaneously applies a single set of instructions to data stored consecutively in an extended-length register. For Intel x86 hardware, streaming SIMD extensions (SSE), advance vector extensions (AVX) and AVX-512 support vector operations on 128, 256 and 512 bit extended registers, respectively. For floating point operations in 64 bit double precision, this amounts to 2-fold, 4-fold and 8-fold theoretical speedups for SSE, AVX and AVX-512, although such performance gains rarely manifest in practice. Whereas many computational statisticians know about multi-core processing, there is little mention of SIMD parallelization in the literature. That said, some R wrapper packages such as RCPPXSIMD and RCPPNT2 (Ushey and Falcou 2016) are becoming available and making it possible for R developers to employ SIMD intrinsics.

We leverage SIMD parallelization by vectorizing or *unrolling* loops within each thread and applying the entire loop body to an entire SIMD extended register at each iteration. For AVX computing in double precision, each iteration of the unrolled loop corresponds to 4 iterations of the original loop. This strategy benefits from efficient reading from and writing to consecutive memory locations and simultaneous evaluation of rate-limiting floating point operations. The use of an instruction-level program profiler reveals that the rate-limiting step in our likelihood calculations is the evaluation of $\exp(\cdot)$ within the inner summation of Eq. (1). Using AVX, for example, one evaluates $\exp(\cdot)$ over four doubles simultaneously and achieves a greater than 2-fold speedup. With less impact on compute performance, we also vectorize the distance calculations between all pairs of location vectors $\mathbf{x}_n$ and $\mathbf{x}_{n'}$ (Holbrook et al. 2019).

2.3.3 Many-core GPUs

GPUs contain hundreds to thousands of cores, but, unlike the independent cores of a CPU, small workgroups of GPU cores must execute the same instruction sets simultaneously though on different data. In this respect, GPU-based parallelization may be thought of as SIMD on a massive scale, leading Nvidia to coin the term SIMT (single instruction, multiple threads) (Lindholm et al. 2008). In this setup, communication between threads within the same workgroup happens extremely quickly via shared on-chip memory, and scheduling a massive number of threads actually hides latencies arising from off-chip memory transactions because of the dynamic and simultaneous loading and off-loading of the many tasks. In part, this is because of the GPU's massively parallel architecture. In part, this is because contemporary general-purpose GPUs have small memory cache but high memory bandwidth, making them ideal tools for performing a massive number of short-lived, cooperative threads.

The likelihood evaluation first involves N independent transformation-reductions, one to obtain each λ_n. We generate $T = N \times B$ threads on the GPU and use work groups of B threads to compute each of the N λ_n. Each thread uses a while-loop across indices n' to compute $\lceil N/B \rceil$ $\lambda_{nn'}$s and keeps a running partial sum. After the threads obtain B partial sums, they work together in a final binary reduction to obtain λ_n. The binary reduction is fast, with $\mathcal{O}(\log B)$ complexity and represents an additional speedup beyond massive parallelization. After computing all N λ_ns, a summation proceeds in the exact same manner. The GPU uses massive parallelization to avoid the cost of the rate-limiting floating point computation in $\exp(\cdot)$. High memory bandwidth allows for fast transfer to and from each working group, and, in turn, each work group shares its own fast access memory that facilitates rapid communication between member threads. We use the Open Computing Language (OpenCL) to write our GPU code. In OpenCL, write functions called kernels, and the library assigns them to each work group separately for parallel execution. To evaluate the likelihood, we write one kernel for the work groups that compute the ℓ_ns and one kernel for those that sum the N ℓ_ns. These details culminate in Algorithm 3. Algorithm 5 is similar to Algorithm 3 and computes the vector of self-excitatory probabilities π_n.

2.4 Software availability

In writing this paper, we have developed hpHawkes https://github.com/suchard-group/hawkes, an open-source R package that enables massively parallel implementations of spatiotemporal Hawkes processes in a big data setting. We have archived a static release of hpHawkes at http://doi.org/10.5281/zenodo.4012745 to aid those who would like to replicate our work. Currently, hpHawkes supports MCMC (Algorithms 1, 2 and 3) for the model described here with the additional capabilities of inferring locations $\mathbf{x}$ and background parameters τ_x and τ_t. In addition to MCMC, hpHawkes supports post-processing of Markov chains to obtain the individual self-excitatory probabilities described in Eq. (2) using Algorithms 4 and 5. This package relies on Rcpp (Eddelbuettel and François 2011) to build and interface with a C++ library that uses OpenCL and TBB frameworks for parallelization on GPUs and CPUs, respectively. We choose to develop with OpenCL because it is both an open-source standard, conforming to our personal support of Open Science (Woelfle et al. 2011), and demonstrates greater portability across devices over its competitors, e.g., CUDA. In making this decision, we have potentially forgone performance gains (Fang et al. 2011), meaning that similar code written in CUDA and based on Algorithms 2–5 could deliver even greater performance increases than those documented below. To enable within-core vectorization, hpHawkes accesses SIMD intrinsics via RcppXsimd (Holbrook et al. 2019), an R package that itself uses Rcpp to access the C++ library Xsimd.

3 Demonstration

In addition to the software we have developed for the purposes of this paper (Sect. 2.4), we have used the R programming language (R Core Team 2019) and the R graphics package ggplot2 (Wickham 2016) to produce the figures and results in the following. The 95% credible intervals we present are highest posterior density intervals that we obtain using the R package coda (Plummer et al. 2006).

3.1 Parallelization

For CPU results, we use a Linux machine with a 10-core Intel Xeon W-2155 processor (3.3 GHz). Each core supports 2 independent threads or logical cores, so the machine reaches a peak performance of 264 gigaflops with double-precision floating point enhanced with AVX vectorization (double that for fused operations such as fused multiply–add). The processor comes with 32 GB DDR4 memory (2667 MHz), 640 KB L1 cache, 10 MB L2 cache and 13 MB L3 cache. For the GPU results, we use an Nvidia Titan V with 5120 CUDA cores (1.2 GHz), achieving 3.1 teraflops peak double-precision floating point performance (again, double this for fused operations). The Titan V comes with 12 GB HBM2 memory, and its 5120 CUDA cores divide into 80 separate *streaming multi-processors* (SM), each consisting of 64 CUDA cores and its own 96 KB L1 cache. Together, all 80 SMs share a single 4.5 MB L2 cache.

Figure 1 and Table 1 show GPU, single-core, multi-core and vectorized processing performances for spatiotemporal

Table 1 Absolute durations and relative speedups of likelihood evaluations for graphics processing unit (GPU) and multi-core advanced vector extensions (AVX) computing relative to single-core AVX computing applied to 75,000 randomly generated observations

Cores vectorization	1			AVX									GPU
	None	SSE		2	4	6	8	10	12	14	16	18	
Duration (s)	117.16	96.94	77.19	40.65	20.73	13.90	10.53	8.59	8.09	7.67	7.28	6.93	0.73
Speedup	0.66	0.80	1	1.90	3.72	5.55	7.33	8.99	9.54	10.07	10.61	11.14	105.54

Single-core implementations without single instruction, multiple data (SIMD) and with streaming SIMD extensions (SSE) occupy the bottom left corner. On our device, non-vectorized, single-core processing reliably achieves between 5-fold and 10-fold speedups over an R-based implementation of the model at hand

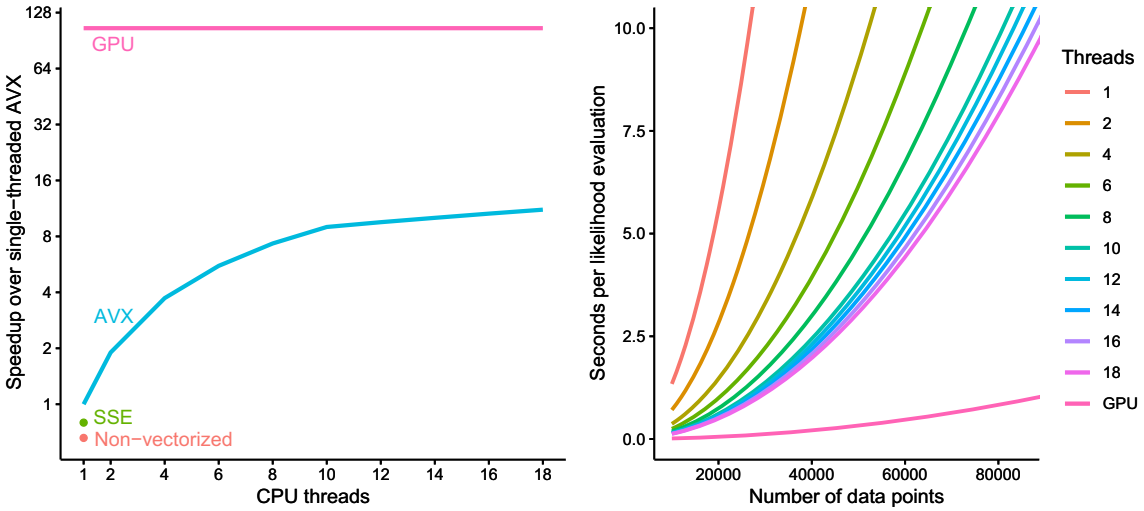

Fig. 1 Spatiotemporal Hawkes process likelihood evaluations. (Left) Speedup of graphics processing unit (GPU) and multi-core advanced vector extensions (AVX) computations relative to single-core AVX computing, all using 75,000 randomly generated observations. Single-core implementations without single instruction, multiple data (SIMD) and with streaming SIMD extensions (SSE) occupy the bottom left corner. (Right) Seconds to compute for both GPU and multi-core AVX processing as a function of data quantity

Hawkes process likelihood evaluations. On the left, we randomly generate $N = 75{,}000$ data points and observe relative speedups over single-threaded AVX processing (77.19 *s*). The GPU implementation (0.73 *s*) is 105× faster, and the 18 thread AVX implementation (6.93 *s*) is 10.4× faster. The roughly 10-fold speedup of the GPU implementation over the 18 thread AVX implementation accords with the former's 3.1 teraflop peak performance relative to the latter's 0.3 teraflop peak performance. On the other hand, the single-threaded AVX implementation is 1.26× and 1.52× faster than the SSE (96.94 *s*) and non-vectorized (117.16 *s*) implementations, respectively. Finally, the GPU implementation is 160× the speed of the single-threaded non-vectorized implementation. On the right, we observe the number of seconds required to perform a single likelihood evaluation for our different implementations as a function of the number of observations, which we let scale from 10,000 data points to 90,000 data points. We compare GPU performance to single- and multi-threaded AVX processing. As expected, all implementations appear to take on a quadratic curve, although one might imagine that the GPU performance has a significantly smaller leading constant.

3.2 Gunshots in Washington, DC

We apply our inference framework to AGLS data generated in Washington D.C. between the years 2006 and 2019 to ascertain the nature of gun violence as a collective phenomenon. Specifically, we wish to determine the extent to which gunfire in D.C. is contagious or diffusionary in nature. We build on, and compare our results to, the analysis of Loeffler and Flaxman (2018), which uses a similar model to that specified in Sect. 2.1. That analysis obtained results from 9000+ data points collected in the years 2010, 2011 and 2012, and the data we use differ from that data two ways. First, we combine datasets located at justicetechlab.org/shotspotter-data (Carr and Doleac 2016, 2018) and https://opendata.dc.gov/datasets to obtain data from 2006 to 2013 and from 2014 to 2019, respectively. Second, these data include the exact second of each event and so have greater temporal precision than that of the previous analysis, which considered data points within the same minute and 100 m radius to be duplicates. In this way, the current dataset is larger because of both greater temporal breadth and greater temporal precision. Like the previous analysis, we consider two datasets, one with all days of the year and one with New Year's Eve, July 4 and surrounding days removed on account of false positives from fireworks and celebratory gunfire. The former ('full+holidays') consists of 85,000+ observations, the latter ('full') 55,000+ observations.

We use Algorithm 1 to generate 4 Markov chains of 10,000 states each and discard the first 1000 states of each chain. Using our GPU and Algorithm 3 to calculate the likelihood within the accept–reject step, total compute time lasts about 4 h for the full analysis and 10 h for the full+holidays analysis. Effective sample sizes are greater than 1700 for all parameters. The top row of Fig. 2 compares posterior inference for lengthscale parameters $1/\omega$, the temporal lengthscale, and h, the spatial lengthscale, between the 'limited' analysis of Loeffler and Flaxman (2018) and our full analysis. We obtain posterior means of 69.5 m (95% CI 68.5, 70.8) and 1.0 min (95% CI 0.98, 1.04) for the two lengthscales, compared to

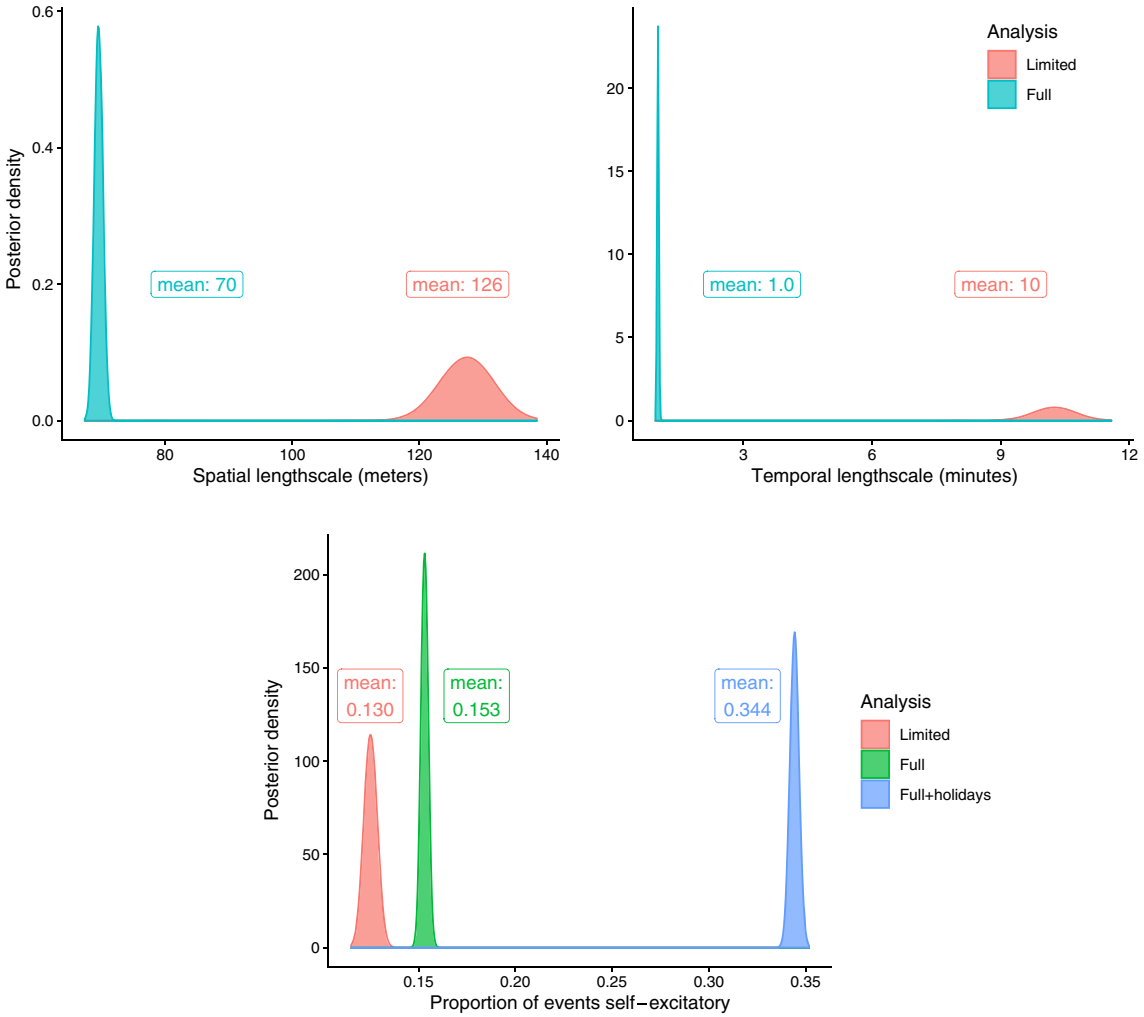

Fig. 2 Posterior distributions of model parameters conditioned on different datasets: 'limited' indicates the 2010 to 2012 analysis of Loeffler and Flaxman (2018) (9000+ observations); 'full' indicates the 2006 to 2019 analysis without New Years and July 4 (55,000+ observations); 'full+holidays' indicates a 2006 to 2019 analysis including New Years and July 4 (85,000+ observations). Larger lengthscales for the limited analysis likely result from thinning of events within the same minute and 100 m range. Both full and limited proportion of events self-excitatory (θ) are within the previously estimated range of 10–18%, whereas that of full + holidays is nowhere near previously estimated ranges

126 m (95% CI 121, 134) and 10 min (95% CI 9.5, 11) for the limited analysis. Both of these results may be expected because the limited analysis removed events within the same minute and 100 m to obtain a thinned dataset about 95% of the original size. As a result, we estimate retaliatory gunfire to occur much sooner after, and closer to, a previous gunshot. To verify this trend, we perform a sensitivity test and remove 8% of the full dataset by considering events within a minute and 100 m from each other to be duplicates. This sensitivity test results in posterior means of 262.4 m (95% CI 253.3, 270.7) and 46.2 min (95% CI 43.6, 48.7) for the two lengthscales. Further sensitivity tests based on 15% and 20% thinned datasets revealed even larger lengthscales. Returning to the full analysis, the posterior variances arising from the full analysis are significantly smaller. This makes sense for two reasons: first, the data conditioned upon are over 5× larger; second, we are considering positive random variables, the variance of which scales with the mean.

In the second row of Fig. 2, we compare posterior densities for parameter θ, which represents the relative weight of the background intensity or the general proportion of events that are self-excitatory in nature. Here, the posterior mean of θ conditioned on the full dataset is 0.153 (95% CI 0.150, 0.156) and larger than the 0.13 (95% CI 0.12, 0.13) of the limited analysis. Again, we attribute this to the lack of data thinning in the full dataset and the resulting greater temporal proximity of gunshots, but we note that both posterior densities nest well within the estimated range of 10–19% for retaliatory homicides of Metropolitan Police Department (2006). On the other hand, the posterior mean of the full+holidays anal-

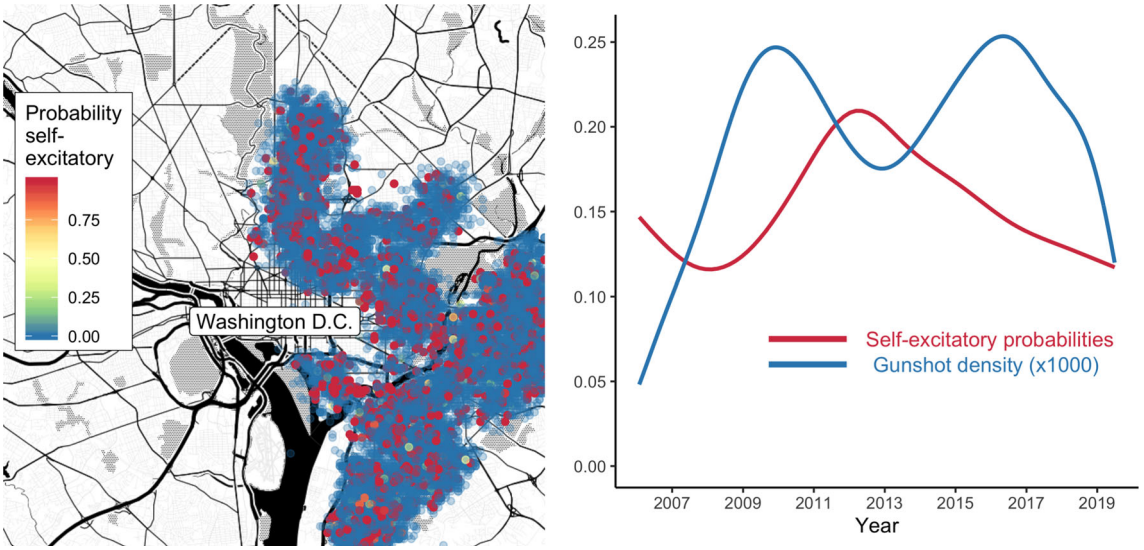

Fig. 3 Posterior means for self-excitatory probabilities π_n (Eq. (2)) in relation to spatial and temporal allocations. (Left) Red indicates a high posterior probability of a gunshot being self-excitatory in nature; blue indicates a low posterior probability. Few yellow points suggests concentration towards values 0 and 1. (Right) We compare smoothing of posterior means for self-excitatory probabilities as a function of time with empirical gunshot trends. A peak in the former around 2013 appears to correspond to a nadir for the latter

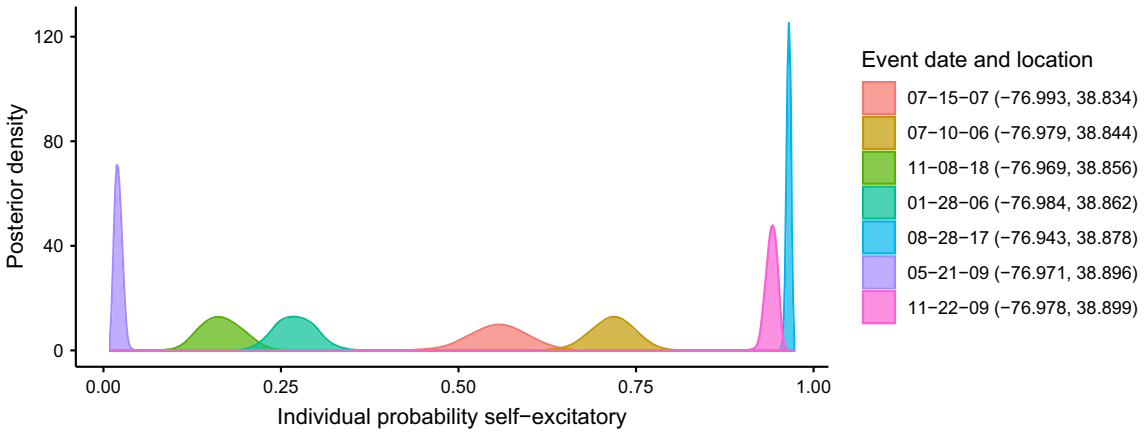

Fig. 4 Posterior distributions for 7 individual probabilities π_n that each gunshot event results from self-excitation. Such distributions may be useful for ascertaining whether specific instances of gun violence are retaliatory in nature. As expected, probabilities close to 0 and 1 vary less. The majority of π_n (not visualized here) resemble point masses extremely close to 0 or 1

ysis is artificially inflated to 0.344 (95% CI 0.340, 0.348) by cascades of fireworks and celebratory gunfire encompassing over one-third of that dataset.

The second half of our analysis considers posterior distributions for the probabilities π_n of each individual gunshot event arising from self-excitation (i.e., being retaliatory) as opposed to the background process. We use our GPU to apply Algorithm 5 to—for storage reasons—a thinned sample of 1000 $\Theta^{(s)}$s to produce 1000 vectors $\pi^{(s)}$ each of length 55,000. In Fig. 3, we visualize the distribution of the posterior means in space and time. On the left, red self-excitatory events distribute fairly evenly among blue background events in Washington D.C., while yellow neutral events barely exist. As a sanity check, the proportion of self-excitatory events seems to roughly coincide with the estimated 0.15 posterior mean of θ. On the right, we smooth posterior self-excitatory probabilities for each event through time, from 2006 to 2009, and compare to the overall gunshot density. In general, the trend in self-excitatory events hits a peak in 2013 of about 20%. This peak coincides with a small dip in the total gunshots for the year 2013, indicating fewer and more closely connected gunfire clusters. Censoring issues make it difficult to interpret relations in these trends near 2006 and 2019. Finally, Fig. 4 presents posterior distributions of probabilities

π_n that 7 individual events are self-excitatory in nature. As may be inferred from Fig. 3's few yellow points, most events cluster close to 0 or 1, resembling a point mass. But many events do provide significant uncertainty, and, as expected, those with posterior mean closer to 0.5 have much greater variability. We believe that figures like Fig. 4 may be useful for crime investigations in determining the retaliatory nature of specific acts of gun violence and quantifying uncertainty in this regard.

R code, data and posterior samples related to the above analysis are available at https://github.com/andrewjholbrook/shot_spotter. To further support replicability, we have archived a static release of the repository at http://doi.org/10.5281/zenodo.4012725. We point out that spatial and temporal censoring bias our results, and we consider corrections for, and modeling of, such bias in a big data context to be a fascinating next step in this line of research.

4 Discussion

Self-excitatory stochastic process models are useful for modeling complex diffusionary and cascading phenomena in multiple scientific disciplines and industrial sectors, but the computational complexity of statistical inference for these models has barred them from applications involving big data. In this paper, we have developed a high-performance statistical computing framework for Hawkes process models that leverages contemporary computational hardware and scales Bayesian inference to more than 85,000 observations. To accomplish this, we have created software for both vectorized multi-core CPU and many-core GPU architecture implementations and made this open-source software freely available online. As a demonstration of the usefulness of this approach, we have applied a spatiotemporal Hawkes process model to the analysis of emerging acoustic gunshot locator systems data recorded in the neighborhoods of Washington D.C. between the years of 2006 and 2019. In this context, Bayesian inference facilitated by our framework provided point estimation and uncertainty quantification of the nature of gun violence as a contagion in American communities. To this end, we have created an additional massively parallel post-processing pipeline to compute probabilities that individual events result from self-excitation based on posterior samples arising from MCMC. These posterior probabilities have proven useful for creating spatial and temporal visualizations that relate self-excitatory gun violence to the Washington D.C. landscape and for quantifying our uncertainty whether individual events are retaliatory in origin. We hope this analysis brings attention to big, complex and emerging AGLS data, the analysis of which might improve scientific understanding of the great American gun violence epidemic.

In the context of this poorly understood epidemic in which many complex models might be posited, fast inference is all the more necessary to facilitate quick candidate model comparison. For example, it is highly doubtful that all self-excitatory action is purely retaliatory in nature: shooting events may consist of multiple shots by the same individual or group. On the other hand, retaliatory shootings may plausibly occur days, weeks or even months after a precipitating event. Thus, it seems that a mixture model employing multiple triggering functions would be appropriate to combine a very short time frame with a slightly longer one or with, perhaps, a much larger time variation (days to months). The reality of multi-shot shooting events, very short-term gunfights and longer term retaliation occurring minutes, hours, days or even months later suggests that additional models to capture these different processes operating over multiple spatial and temporal scales will be needed. This only reinforces the need for fast computation, which will support selection between more complex models as well as comparisons to the simpler ones already in the literature.

We will also extend our high-performance computing framework to other generalizations of the Hawkes process such as marked Hawkes processes and mutually exciting point processes. The former have been effective for modeling Earthquakes (here, the mark is the tremor's score on the Richter scale), the latter for modeling dependencies between neurons. For these efforts to succeed and enjoy maximal impact, we must scale Bayesian inference for such point process models to millions of observations, and we believe that computational tools that accomplish fine-grained parallelization (e.g., tensor processing units and bigger, faster GPUs) will accomplish more than multi-processor approaches that fail to overcome inherent latency and communication bottlenecks. Nonetheless, we are also interested in developing inference frameworks that share computational resources between both CPU and GPU simultaneously. For scalable Bayesian inference, all computing tools and computational hardware must be on the table. After all, Washington D.C. is only one city of at least 40 for which AGLS data have come available in the last decade: American gunfire data are big data, indeed.

Acknowledgements The research leading to these results has received funding through National Institutes of Health Grant U19 AI135995 and National Science Foundation Grant DMS1264153. AJH is supported by NIH Grant K25AI153816. We gratefully acknowledge support from Nvidia Corporation with the donation of parallel computing resources used for this research.

References

Allaire, J., Francois, R., Ushey, K., Vandenbrouck, G., Geelnard, M.: Intel: RcppParallel: Parallel Programming Tools for 'Rcpp'. R package version 4.3.19 (2016)

Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, pp. 483–485 (1967)

Beam, A.L., Ghosh, S.K., Doyle, J.: Fast Hamiltonian Monte Carlo using GPU computing. J. Comput. Graph. Stat. **25**, 536–548 (2016)

Bjerregaard, B., Lizotte, A.J.: Gun ownership and gang membership. J. Crim. L. Criminol. **86**, 37 (1995)

Carr, J., Doleac, J.L.: The geography, incidence, and underreporting of gun violence: new evidence using shotspotter data. In: Incidence, and Underreporting of Gun Violence: New Evidence Using Shotspotter Data (2016)

Carr, J.B., Doleac, J.L.: Keep the kids inside? Juvenile curfews and urban gun violence. Rev. Econ. Stat. **100**, 609–618 (2018)

Centers for Disease Control and Prevention: Centers for Disease Control and Prevention, National Center for Health Statistics. Underlying Cause of Death 1999–2018 on CDC WONDER Online Database, released in 2020. Data are from the Multiple Cause of Death Files, 1999–2018, as compiled from data provided by the 57 vital statistics jurisdictions through the Vital Statistics Cooperative Program (2020). Accessed wonder.cdc.gov/ucd-icd10.html

Chavez-Demoulin, V., McGill, J.: High-frequency financial data modeling using Hawkes processes. J. Bank. Finance **36**, 3415–3426 (2012)

Choi, E., Du, N., Chen, R., Song, L., Sun, J.: Constructing disease network and temporal progression model via context-sensitive Hawkes process. In: 2015 IEEE International Conference on Data Mining, pp. 721–726. IEEE (2015)

Daley, D.J.: An Introduction to the Theory of Point Processes: Elementary Theory of Point Processes. Springer, Berlin (2003)

Daley, D.J., Vere-Jones, D.: An Introduction to the Theory of Point Processes: Volume II: General Theory and Structure. Springer, Berlin (2007)

Eddelbuettel, D., François, R.: Rcpp: Seamless R and C++ integration. J. Stat. Softw. **40**, 1–18 (2011)

Embrechts, P., Liniger, T., Lin, L.: Multivariate Hawkes processes: an application to financial data. J. Appl. Probab. **48**, 367–378 (2011)

Fang, J., Varbanescu, A.L., Sips, H.: A comprehensive performance comparison of cuda and opencl. In: 2011 International Conference on Parallel Processing, pp. 216–225. IEEE (2011)

Federal Bureau of Investigation: Crime in the u.s. (2005). Accessed www2.fbi.gov/ucr/05cius/data/table_05.html

Flaxman, S.R.: Machine Learning in Space and Time. Ph.D. thesis, Carnegie Mellon University (2015)

Gelman, A., Roberts, G.O., Gilks, W.R., et al.: Efficient metropolis jumping rules. Bayesian Stat. **5**, 42 (1996)

Grisales, C.: From Border Security to Tobacco Age, Both Parties Tout Key Wins in Spending Deal. NPR. Accessed (2019). www.npr.org/2019/12/16/788506571/border-wall-to-tobacco-age-both-parties-tout-key-wins-in-spending-deal

Haario, H., Saksman, E., Tamminen, J., et al.: An adaptive metropolis algorithm. Bernoulli **7**, 223–242 (2001)

Hardiman, S.J., Bercot, N., Bouchaud, J.-P.: Critical reflexivity in financial markets: a Hawkes process analysis. Eur. Phys. J. B **86**, 442 (2013)

Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**, 97–109 (1970)

Hawkes, A.G.: Point spectra of some mutually exciting point processes. J. R. Stat. Soc. Ser. B Methodol. **33**, 438–443 (1971a)

Hawkes, A.G.: Spectra of some self-exciting and mutually exciting point processes. Biometrika **58**, 83–90 (1971b)

Hawkes, A.: Spectra of some mutually exciting point processes with associated variables. Stoch. Point Process. 261–271 (1972)

Hawkes, A.: Cluster models for earthquakes-regional comparisons. Bull. Int. Stat. Inst. **45**, 454–461 (1973)

Hawkes, A.G.: Hawkes processes and their applications to finance: a review. Quant. Finance **18**, 193–198 (2018)

Holbrook, A., Lemey, P., Baele, G., Dellicour, S., Brockmann, D., Rambaut, A., Suchard, M.: Massive parallelization boosts big Bayesian multidimensional scaling. arXiv preprint arXiv:1905.04582 (2019)

Kelly, J.D., Park, J., Harrigan, R.J., Hoff, N.A., Lee, S.D., Wannier, R., Selo, B., Mossoko, M., Njoloko, B., Okitolonda-Wemakoy, E., et al.: Real-time predictions of the 2018–2019 ebola virus disease outbreak in the democratic republic of the congo using hawkes point process models. Epidemics **28**, 100354 (2019)

Kim, H.: Spatio-temporal Point Process Models for the Spread of Avian Influenza Virus (H5N1). Ph.D. thesis UC Berkeley (2011)

Laub, P.J., Taimre, T., Pollett, P.K.: Hawkes processes. arXiv preprint arXiv:1507.02822 (2015)

Lee, A., Yau, C., Giles, M.B., Doucet, A., Holmes, C.C.: On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. J. Comput. Graph. Stat. **19**, 769–789 (2010)

Linderman, S., Adams, R.: Discovering latent network structure in point process data. In: International Conference on Machine Learning, pp. 1413–1421 (2014)

Linderman, S.W., Wang, Y., Blei, D.M.: Bayesian inference for latent Hawkes processes. Adv. Neural Inf. Process. Syst. (2017)

Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: Nvidia tesla: a unified graphics and computing architecture. IEEE Micro **28**, 39–55 (2008)

Loeffler, C., Flaxman, S.: Is gun violence contagious? A spatiotemporal test. J. Quant. Criminol. **34**, 999–1017 (2018)

Mares, D., Blackburn, E.: Evaluating the effectiveness of an acoustic gunshot location system in St. Louis, MO. Polic. J. Policy Pract. **6**, 26–42 (2012)

Mei, H., Eisner, J.M.: The neural Hawkes process: A neurally self-modulating multivariate point process. In: Advances in Neural Information Processing Systems, pp. 6754–6764 (2017)

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**, 1087–1092 (1953)

Metropolitan Police Department: Juvenile and Adult Homicide in the District of Columbia—2001–2005 (2006)

Meyer, S., Held, L., et al.: Power-law models for infectious disease spread. Ann. Appl. Stat. **8**, 1612–1639 (2014)

Mohler, G.: Marked point process hotspot maps for homicide and gun crime prediction in Chicago. Int. J. Forecast. **30**, 491–497 (2014)

National Research Council: Firearms and Violence: A Critical Review. National Academies Press (2005)

National Research Council: Priorities for Research to Reduce the Threat of Firearm-Related Violence. National Academies Press (2013)

Ogata, Y.: Statistical models for earthquake occurrences and residual analysis for point processes. J. Am. Stat. Assoc. **83**, 9–27 (1988)

Park, J., Schoenberg, F.P., Bertozzi, A.L., Brantingham, P.J.: Investigating Clustering and Violence Interruption in Gang-Related Violent Crime Data Using Spatial–Temporal Point Processes with Covariates (2019)

Petho, A., Fallis, D., Keating, D.: Shotspotter Detection System Documents 39,000 Shooting Incidents in the District. Washington Post (2013). Accessed www.washingtonpost.com/investigations/

Plummer, M., Best, N., Cowles, K., Vines, K.: Coda: convergence diagnosis and output analysis for MCMC. R News **6**, 7–11 (2006)

R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing Vienna, Austria (2019)

Rasmussen, J.G.: Bayesian inference for Hawkes processes. Methodol. Comput. Appl. Probab. **15**, 623–642 (2013)
Ratcliffe, J.H., Rengert, G.F.: Near-repeat patterns in Philadelphia shootings. Secur. J. **21**, 58–76 (2008)
Reinders, J.: Intel Threading Building Blocks, 1st edn. O'Reilly & Associates Inc, Sebastopol (2007)
Reinhart, A., Greenhouse, J.: Self-exciting point processes with spatial covariates: modelling the dynamics of crime. J. R. Stat. Soc. Ser. C **67**, 1305–1329 (2018)
Reinhart, A., et al.: A review of self-exciting spatio-temporal point processes and their applications. Stat. Sci. **33**, 299–318 (2018)
Rizoiu, M.-A., Mishra, S., Kong, Q., Carman, M., Xie, L.: Sir–Hawkes: linking epidemic models and Hawkes processes to model diffusions in finite populations. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web International World Wide Web Conferences Steering Committee, pp. 419–428 (2018)
Roberts, G.O., Rosenthal, J.S.: Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. J. Appl. Probab. **44**, 458–475 (2007)
Roberts, G.O., Rosenthal, J.S.: Examples of adaptive MCMC. J. Comput. Graph. Stat. **18**, 349–367 (2009)
Rubin, R.: Tale of 2 agencies: CDC avoids gun violence research but NIH funds it. JAMA **315**, 1689–1692 (2016)
Schoenberg, F.P.: Facilitated estimation of etas. Bull. Seismol. Soc. Am. **103**, 601–605 (2013)
Showen, R.: Operational gunshot location system. In: Surveillance and Assessment Technologies for Law Enforcement, Vol. 2935 International Society for Optics and Photonics, pp. 130–139 (1997)
Suchard, M., Rambaut, A.: Many-core algorithms for statistical phylogenetics. Bioinformatics **25**, 1370–1376 (2009)
Suchard, M., Wang, Q., Chan, C., Frelinger, J., Cron, A., West, M.: Understanding GPU programming for statistical computation: studies in massively parallel massive mixtures. J. Comput. Graph. Stat. **19**, 419–438 (2010a)
Suchard, M.A., Holmes, C., West, M.: Some of the what?, why?, how?, who? and where? of graphics processing unit computing for Bayesian analysis. Bull. Int. Soc. Bayesian Anal. **17**, 12–16 (2010b)
Truccolo, W.: From point process observations to collective neural dynamics: nonlinear Hawkes process glms, low-dimensional dynamics and coarse graining. J. Physiol. Paris **110**, 336–347 (2016)
Ushey, K., Falcou, J.: RcppNT2: 'Rcpp' Integration for the 'NT2' Scientific Computing Library. R package version 0.1.0 (2016)
Wadman, M.: Firearms research: the gun fighter. Nat. News **496**, 412 (2013)
Warne, D.J., Sisson, S.A., Drovandi, C.: Acceleration of expensive computations in Bayesian statistics using vector operations (2019). arXiv preprint arXiv:1902.09046
White, G., Porter, M.D.: GPU accelerated MCMC for modeling terrorist activity. Comput. Stat. Data Anal. **71**, 643–651 (2014)
Wickham, H.: ggplot2: Elegant Graphics for Data Analysis. Springer, New York (2016)
Woelfle, M., Olliaro, P., Todd, M.H.: Open science is a research accelerator. Nat. Chem. **3**, 745–748 (2011)
Yang, S.-H., Zha, H.: Mixture of mutually exciting processes for viral diffusion. In: International Conference on Machine Learning, pp. 1–9 (2013)
Zhou, H., Lange, K., Suchard, M.: Graphics processing units and high-dimensional optimization. Stat. Sci. **25**, 311–324 (2010)
Zhuang, J., Ogata, Y., Vere-Jones, D.: Analyzing earthquake clustering features by using stochastic reconstruction. J. Geophys. Res. Solid Earth (2004). https://doi.org/10.1029/2003JB002879