ACCELERATING MAXIMUM LIKELIHOOD ESTIMATION FOR HAWKES POINT PROCESSES

Ce Guo and Wayne Luk

Department of Computing,
Imperial College London
London, SW7 2AZ, UK
{ce.guo10, w.luk}@imperial.ac.uk

ABSTRACT

Hawkes processes are point processes that can be used to build probabilistic models to describe and predict occurrence patterns of random events. They are widely used in high-frequency trading, seismic analysis and neuroscience. A critical numerical calculation in Hawkes process models is parameter estimation, which is used to fit a Hawkes process model to a data set. The parameter estimation problem can be solved by searching for a parameter set that maximises the log-likelihood. A core operation of this search process, the log-likelihood evaluation, is computationally demanding if the number of data points is large. To accelerate the computation, we present a log-likelihood evaluation strategy which is suitable for hardware acceleration. We then design and optimise a pipelined engine based on our proposed strategy. In the experiments, an FPGA-based implementation of the proposed engine is shown to be up to 72 times faster than a single-core CPU, and 10 times faster than an 8-core CPU.

1. INTRODUCTION

Hawkes processes [1] are point processes that can be used to build probabilistic models to describe and predict occurrence patterns of random events. The study of Hawkes process models has attracted the attention of researchers and practitioners from various areas including high-frequency trading [2, 3], seismology [4, 5] and neuroscience [6, 7].

Hawkes process models are parametric. To model the occurrence pattern of a random event with a Hawkes process model, one needs to estimate the parameters using historical occurrence data of the event. The estimated model can be used to predict the occurrence frequency in the near future.

The parameter estimation problem of Hawkes processes can be solved by maximum likelihood estimation [8, 9]. This is a challenging task because the core operation, the log-likelihood evaluation, is computationally demanding if the number of points in the data set is large. Unfortunately, industrial-scale data sets are usually large and their sizes have been growing swiftly in recent years. For instance, financial markets order placement frequencies have increased in orders of magnitude over the last several years and so has the scale of the models used to describe this activity. Moreover, it is desirable to invoke parameter estimation frequently in order to keep the model up-to-date. The conflict between the data size and computational efficiency is especially serious in time critical problems such as high-frequency trading and optimal order execution [10, 11].

While systems based on point process models can benefit from the speed, simplicity and power efficiency of hardware implementations, hardware acceleration of point processes is not well-studied. To the best of our knowledge, our work is the first to cover hardware acceleration of model fitting of point processes. Our key contributions are as follows.

- We present a log-likelihood evaluation strategy by setting up incremental computation rules and eliminating complex data dependency. This strategy is particularly suitable for hardware acceleration.
- We design a pipelined hardware architecture based on our strategy. We then propose a series of optimisations for our hardware architecture to avoid pipeline stalls, to promote parallelism among multiple iterations, to reduce memory bandwidth consumption, and to remove redundant logarithm evaluation units.
- We implement our design in a commercial FPGA acceleration platform, and compare its performance with a CPU implementation on a single core and eight cores.

The rest of the paper is organised as follows. Section 2 briefly describes point processes, Hawkes processes and the maximum likelihood estimation problem. Section 3 presents a log-likelihood evaluation strategy for hardware acceleration. Section 4 describes the design and optimisations of a pipelined hardware architecture based on the proposed strategy. Section 5 provides experimental results about an implementation of our hardware design, and explains experimental observations. Section 6 provides a brief conclusion, and discusses possible future work.

2. BACKGROUND

In this section we provide a short introduction to Hawkes processes. We first illustrate related concepts including point processes, counting processes, intensities and Hawkes processes. Then we briefly discuss the maximum likelihood estimation problem.

2.1. Point Processes and Hawkes Processes

A sequence of random variables, $\{t_i\} = (t_1, t_2, t_3, \dots)$, is a *point process* if and only if $t_i > 0$ and $t_i < t_{i+1}$ for all $i \in \mathbb{N}^+$. A point process is typically used to describe the occurrences of a repeatable event. Each entry of the process is the time of an occurrence of the event. The counting process of a point process, $C(t)$, is defined by

$$C(t) = \sum_{i \in \mathbb{N}^+} 1_{t_i \le t} \tag{1}$$

Using the counting process, we may describe the frequency of the occurrences using the intensity function, which is defined by

$$\lambda(t) = \lim_{h \downarrow 0} \frac{1}{h} \mathbb{P}(C(t+h) - C(t) > 0) \tag{2}$$

where $\mathbb{P}(X)$ is the probability of random event X. In other words, the intensity of a point process taken at a fixed point in time, t, is the instantaneous rate at time t with which the event occurs. A point process $\{t_i\}$ is a *Hawkes process* if the intensity function $\lambda(\cdot)$ satisfies

$$\lambda(t) = \lambda^{\perp} + \int_0^t g(t-w) \mathrm{d}C_{m'}(w) \tag{3}$$

where $\lambda^{\perp}$ is a constant; $C(\cdot)$ is the counting process of $\{t_i\}$ and $g(\cdot)$ is a response function which needs to be integrable and positive. In this study, we focus on the exponential response function [1] defined by

$$g(x) = \alpha e^{-\beta x} \tag{4}$$

where α and β are constant parameters. With this response function, the intensity function can be written as

$$\lambda(t) = \lambda^{\perp} + \alpha \sum_{k'=1}^{K(t)} e^{-\beta(t - t_{k'})} \tag{5}$$

where $K(t)$ is the number of occurrences by time t. The feature that makes the Hawkes process useful for making predictions and modelling data is that its intensity (a) is stochastic, (b) changes in time and (c) depends on the history of the process.

Hawkes processes models are parametric. A parameter set for a Hawkes process with an exponential response function is a triplet in the form $\theta = \langle \lambda^{\perp}, \alpha, \beta \rangle$. Note that the intensity function $\lambda(t)$ depends on the parameter set θ. We write $\lambda(t)$ instead of $\lambda(t|\theta)$ for brevity.

2.2. Parameter Estimation for Hawkes Processes

Parameter estimation for Hawkes process models is the process to find a reasonable parameter set to fit a model to a data set. This process is the critical part in various applications that involve Hawkes process models. For instance, Hawkes processes can be used to predict the occurrence frequency of a random event in the near future. The first step to perform the prediction is to fit a model using historical data.

Parameter estimation can be achieved by maximum likelihood estimation [9] where the parameter set is estimated by the following estimator

$$\theta_{\mathrm{MLE}} = \mathrm{argmax}_{\theta} L(\theta) \tag{6}$$

where $\mathrm{argmax}_{\mathrm{x}}(f(x))$ is the set of argument x that maximises function $f(x)$; $L(\theta)$ is the log-likelihood for the parameter θ is defined by

$$L(\theta) = \int_0^T \log \lambda(t) \mathrm{d}C(t) - \int_0^T \lambda(t) \mathrm{d}t \tag{7}$$

Given a data set with K observations $\mathcal{D} = (t_1, t_2, \dots, t_K)$, the log-likelihood function can be written in a computable form

$$L(\theta) = \sum_{k=1}^{K} r(k) - \lambda^{\perp} T + \sum_{k=1}^{K} h(k) \tag{8}$$

where

$$r(k) = \log \lambda(t_k) = \log(\lambda^{\perp} + \alpha \sum_{k'=1}^{k-1} e^{-\beta(t_k - t_{k'})}) \tag{9}$$

$$h(k) = \frac{\alpha}{\beta}(e^{-\beta(T - t_k)} - 1) \tag{10}$$

where T is the time of termination of sampling. In this study, we take $T = t_K$ following [8]. The maximum likelihood estimation technique is proved to be asymptotically normal with optimal convergence rate and optimal variance [2]. However, there is no close-form expression for θ_{MLE} in Equation 6, therefore one has to search over the parameter space for a parameter set that maximises the log-likelihood [8].

The search of parameters can be conducted by a general-purpose optimisation algorithm. No matter what algorithm is used, it needs to evaluate log-likelihood frequently. This evaluation process can be computationally expensive if the number of data points is large. Practically, large data sets are often used for estimation accuracy. As a result, the log-likelihood evaluation usually dominates the execution time of the parameter search. In this study, we accelerate the parameter estimation process by designing a pipelined hardware architecture for log-likelihood evaluation.

The idea of accelerating log-likelihood evaluation for statistical models using FPGAs has been proposed [12, 13]. However, our acceleration solutions for Hawkes point process is significantly different from existing work because the evaluation algorithms for different problem settings do not share a common computational pattern.

3. ACCELERATION STRATEGY

In this section, we present a log-likelihood evaluation strategy for Hawkes process models which is suitable for reconfigurable hardware. We first explain the mathematical background of the strategy, and then discuss a method for eliminating complex data dependencies.

3.1. Incremental Log-likelihood Evaluation

For hardware efficiency, we compute the log-likelihood function with sequential data access and simple data dependency. Given a data set $\mathcal{D} = (t_1, t_2, \ldots, t_K)$, we define

$$Q(k) = \sum_{k=1}^{K} \big(r(k) + h(k)\big) \tag{11}$$

From this definition and Equation 8, we express the log-likelihood using $Q(K)$ by

$$L(\theta) = Q(K) - \lambda^{\perp} T \tag{12}$$

$Q(k)$ can be computed recursively by

$$Q(k) = \begin{cases} 0 & \text{if } k = 0 \\ Q(k-1) + r(k) + h(k) & \text{otherwise} \end{cases} \tag{13}$$

where $r(k)$ and $h(k)$ are defined in Equation 9 and 10 respectively.

We may calculate $h(k)$ using Equation 10 directly. The computation is obvious because $h(k)$ only depends on the parameter set and the data point t_k. However, it may not be wise to compute $r(k)$ using Equation 9 because $r(k)$ depends on $t_1, t_2, \ldots, t_{k-1}$. This data dependency leads to high time complexity and makes the computation inappropriate for hardware implementation. With respect to this issue, we present an alternative way to compute $r(k)$ in the next subsection.

3.2. Recursive Computation for $r(k)$

The exponential response function provides the opportunity to eliminate the complex data dependency in the computation of $r(k)$. We present one possible strategy to achieve this by applying the log-likelihood decomposition technique in [9] to our response function in Equation 4. Let

$$\phi(k) = \sum_{k'=1}^{k-1} e^{-\beta(t_k - t_{k'})} \tag{14}$$

Then the summation operation in Equation 9 can be replaced by $\phi(k)$,

$$r(k) = \log \lambda(t_k) = \log \big(\lambda^{\perp} + \alpha\phi(k)\big) \tag{15}$$

If $\phi(k)$ can somehow be obtained, then $r(k)$ can be computed easily. Therefore we focus on calculating $\phi(k)$. When $k = 1$, we have $\phi(1) = 0$. When $k > 1$, from Equation 14, $\phi(k)$ can be written as

$$\phi(k) = \sum_{k'=1}^{k-2} e^{-\beta(t_k - t_{k'})} + e^{-\beta(t_k - t_{k-1})} \tag{16}$$

We explore relationship between the case where $k' = k$ and the case where $k' = k - 1$ by performing the following transformation on $\phi(k)$:

$$\phi(k) = e^{-\beta(t_k - t_{k-1})} \sum_{k'=1}^{k-2} e^{-\beta(t_{k-1} - t_{k'})} + e^{-\beta(t_k - t_{k-1})} \tag{17}$$

Substitute Equation 14 to Equation 17, we obtain:

$$\phi(k) = e^{-\beta(t_k - t_{k-1})}\big(\phi(k-1) + 1\big) \tag{18}$$

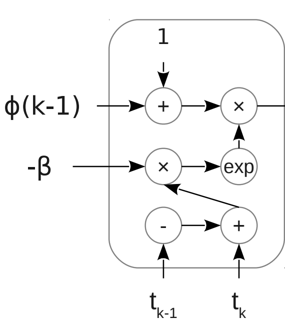
Therefore, $r(k)$ and $\phi(k)$ can be computed given $\phi(k-1)$, which eliminates the data dependency in Equation 9. This is particularly useful for hardware design.

4. HARDWARE DESIGN AND OPTIMISATION

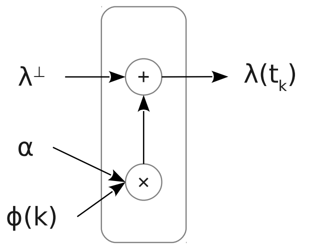
In this section, we map the strategy derived in the previous section to a pipelined hardware architecture. We first describe three hardware modules and discuss possible ways to combine them to compute $Q(k)$ efficiently.

4.1. Basic Architecture

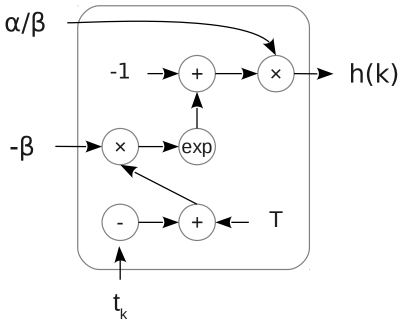
We propose three arithmetic modules: the ϕ module, λ module and h module. Their structures are shown in Fig. 1. The ϕ module is designed based on Equation 18. The base case of the function when $k = 1$ is not reflected in the hardware. By supplying $\phi(0) = -1$ as the input to a ϕ module, we can obtain $\phi(1) = 0$. The λ module is designed based on Equation 15. We do not take logarithms for the purpose of optimisation, which will be explained in Section 4.3. The h module is based on Equation 10. One input to the h module, α/β, is computed in the host computer to avoid implementing division in the reconfigurable device.



(a) ϕ Module



(b) λ Module



(c) h Module

Fig. 1. Basic Arithmetic Modules

An elementary combination of the basic modules, the Q module, is shown in Fig. 2. This module is designed based on Equation 13. It computes $Q(k)$ and $\phi(k)$ by taking $Q(k-1)$, $\phi(k-1)$ and model parameters as the inputs.

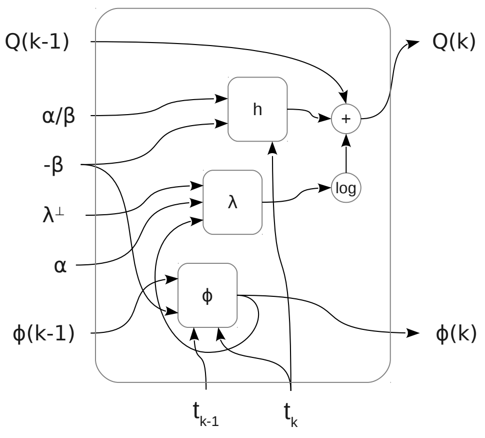


Fig. 2. A Simple Q Module

Due to the latency of the module, it is impossible to use an output of $Q(k)$ as the input immediately for the next iteration. Therefore, evaluating the log-likelihood for a single parameter set will result in pipeline stalls. To ensure efficiency, we apply c-slow retiming [14] to process a batch of P parameter sets together to keep the pipeline busy. More specifically, a data point t_k is loaded from the data stream at the beginning of every P cycles. $Q(k)$ for parameter set $1 \ldots P$ are computed based on t_k within P cycles before the next data point is loaded.

4.2. Parallelism over Multiple Iterations

When multiple consecutive data points are available, it is possible to combine the basic modules to compute multiple iterations. From Equation 13, when $k > 1$ we have

$$Q(k) = Q(k-W-1) + \sum_{w=1}^{W} r(k-w) + \sum_{w=1}^{W} h(k-w) \tag{19}$$

According to this equation, we design a Q module to compute $Q(k)$ using $Q(k-W)$ and consecutive $(W+1)$ data points. For example, an architecture which takes 4 consecutive data points is shown in Fig. 3. The inputs of model parameters are omitted for simplicity.

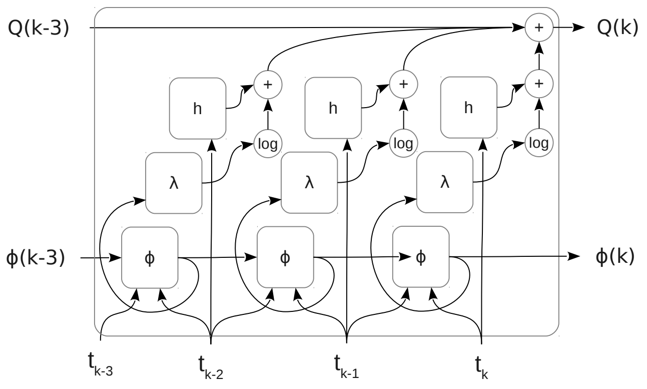


Fig. 3. Q Module Handling Multiple Iterations

If the pipeline is kept busy, W iterations of the computation of $Q(k)$ for a parameter set can be performed in a single cycle. As a result, we may boost the performance by integrating more λ, ϕ and h modules to handle more iterations. A Q module handling W iterations requires $(W+1)$ consecutive data points. W out of the $(W+1)$ data points need to be collected from the data stream. When processing a batch of P parameter sets, these inputs need to be updated every P cycles. Suppose the update occurs at the beginning of cycle u, we collect the W data points from cycle $(u-W)$ to $(u-1)$ and save the data points in a buffer. With this buffering scheme, only one data point is accessed in each cycle, which results in low memory bandwidth requirement.

4.3. Logarithm Evaluation Reduction

The architecture shown in Fig. 3 needs W logarithm evaluation units to compute W iterations in one cycle. However, these units are expensive in terms of computational resources. It is desirable to reduce the number of logarithm evaluation units. From Equation 19,

$$\sum_{w=1}^{W} r(k-w) = \sum_{w=1}^{W} \log \lambda(t_{k-w}) = \log \prod_{w=1}^{W} \lambda(t_{k-w}) \tag{20}$$

Therefore, we replace the W logarithm evaluation units by a single one and $(W-1)$ multipliers, which reduces the overall resource consumption. For instance, an optimised version of the architecture in Fig. 3 is shown in Fig. 4. The effect of this optimisation depends on the design of logarithm evaluation units. For example, in our experimental implementation, the number of iterations in one cycle, W, increases from 10 to 12 with this optimisation.

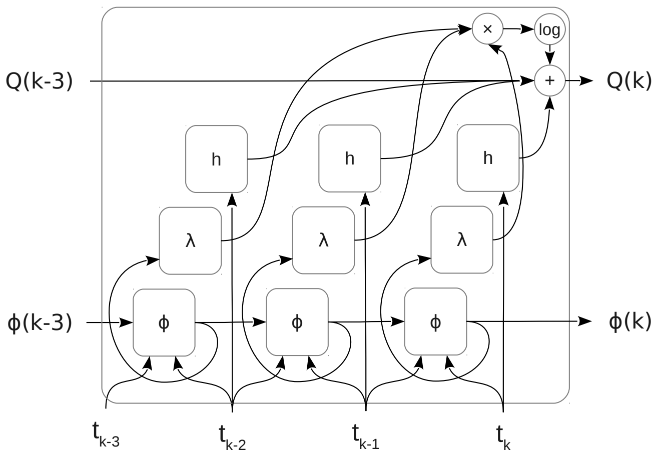


Fig. 4. Q Module with Logarithm Evaluation Reduction

5. EXPERIMENTAL EVALUATION

We implement the proposed architecture with all the discussed optimisations in an FPGA-based acceleration platform and test its performance using data sets with various sizes. In this section, we first describe the experimental settings and then discuss experimental results.

5.1. Experimental Settings

We implement our hardware design on a Maxeler MAX3 acceleration system, which is equipped with a Xilinx Virtex-6 V6-SXT475 FPGA. It communicates with the host computer via a PCI-Express interface. The clock frequency of the FPGA is set to 100MHz. The hardware is described in the MaxJ language and compiled to VHDL using Maxeler MaxCompiler. The Q module we implemented computes $W = 12$ iterations for $Q(k)$. This parameter is selected to maximise hardware resource usage for better performance. To keep the pipeline busy, we evaluate $P = 1024$ parameters in a batch to avoid pipeline stalls.

We also build an implementation that runs exclusively on the CPU platform using the OpenMP library. Both the host code for the FPGA implementation and the pure CPU implementation are written in the C programming language and compiled using the Intel C compiler with the highest compiling optimisation. Both systems are invoked in a server with an Intel Xeon X5650 CPU running at 2.67GHz.

The IEEE single precision floating point numbers are used throughout the FPGA and CPU implementations except for the inputs of the logarithm evaluation unit in FPGA, where double precision floating point numbers are employed for better accuracy. A log-likelihood value computed by the FPGA implementation may slightly differ from the one computed by the CPU implementation, but the relative difference is controlled below 0.1%.

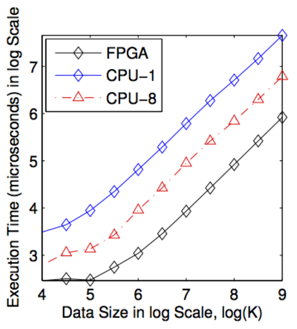
The Nelder-Mead simplex algorithm is used as the search algorithm for parameter sets. We first run the algorithm in the CPU platform and record all the log-likelihood queries. We then execute this set of queries on the tested systems. The performance is measured by the execution time for each log-likelihood query. To make a fair comparison, the latency overhead for CPU-FPGA communication is included in the experimental results. To make the results less dependent on a particular search algorithm, the execution time of the Nelder-Mead simplex algorithm is excluded. We have noticed that an inappropriate search algorithm in the host computer may incur unnecessary overhead and limit the overall performance. This issue will be discussed in a future publication.

5.2. Results and Discussion

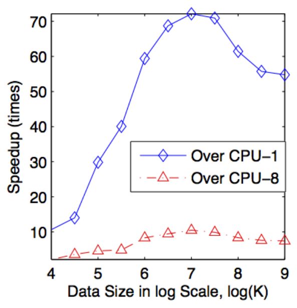
The experimental results are plotted in Fig. 5. The execution times for a query are recorded in Fig. 5(a). The speedup of the FPGA implementation over the CPU implantation on a single core and eight cores are presented in Fig. 5(b). We record the data size in log scale with base 10 in both figures. To reflect the trend of the increment of the execution time, we also plot the results in log scale in Fig. 5(a).

The execution times of both the CPU and FPGA implementation increase approximately linearly as the data size grows. The CPU implementation works well on eight cores. It is around 7.1 times faster than a single core.

When data size $K \leq 10^7$, the speedup of the FPGA implementation over the CPU implementation increases as the size of the data set grows. This is because, while the execution time spent on the pipeline grows linearly with the size of the data set, the overhead including the hardware initiali-



(a) Execution Time



(b) Speedup

Fig. 5. Experimental Results

sation time and post-processing time does not increase. The peak speedup is 72 times and 10 times over the CPU implementation on a single core and eight cores respectively.

When $K > 10^7$, the speedup of the FPGA implementation decreases slightly and stabilises at around 55 times over a single core. We find that the increment of execution time of the FPGA implementation follows a linear pattern but that of the CPU implementation is slightly below a linear trend when $K > 10^7$. We are unsure about the reason behind this observation, but a possible explanation is that the compiler optimised the code with respect to large data.

6. CONCLUSION

This paper presents a hardware architecture for log-likelihood evaluation to accelerate parameter estimation for Hawkes point processes. We set up incremental computation rules and eliminate complex data dependency, resulting in a log-likelihood evaluation strategy particularly suitable for hardware acceleration. We design a pipelined hardware architecture based on the proposed strategy and optimise it from various perspectives. Our experimental system implemented in a Virtex-6 V6-SXT475 FPGA achieves up to 72 times faster than a single-core CPU, and up to 10 times faster than an 8-core CPU.

We consider our work a first step towards hardware acceleration of Hawkes process modelling. One possible future work is to design parameter estimation engines for multivariate Hawkes processes which can be used to capture the relationship among multiple random events. Another possible future work is to integrate this acceleration solution with predictive systems to forecast future occurrences of events in situations such as high-frequency trading.

7. ACKNOWLEDGEMENTS

This work is supported in part by the China Scholarship Council, by the European Union Seventh Framework Programme under grant agreement number 257906, 287804 and 318521, by UK EPSRC, by Maxeler University Programme, and by Xilinx.

8. REFERENCES

[1] A. Hawkes, “Spectra of some self-exciting and mutually exciting point processes,” *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971.

[2] E. Bacry, S. Delattre, M. Hoffmann, and J. Muzy, “Modelling microstructure noise with mutually exciting point processes,” *Quantitative Finance*, vol. 13, no. 1, pp. 65–77, 2013.

[3] C. G. Bowsher, “Modelling security market events in continuous time: Intensity based, multivariate point process models,” *Journal of Econometrics*, vol. 141, no. 2, pp. 876–912, 2007.

[4] Y. Y. Kagan, “Statistical distributions of earthquake numbers: consequence of branching process,” *Geophysical Journal International*, vol. 180, no. 3, pp. 1313–1328, 2010.

[5] Y. Ogata and K. Katsura, “Point-process models with linearly parametrized intensity for application to earthquake data,” *Journal of Applied Probability*, vol. 23, pp. 291–310, 1986.

[6] R. Dahlhaus, M. Eichler, and J. Sandkühler, “Identification of synaptic connections in neural ensembles by graphical models,” *Journal of neuroscience methods*, vol. 77, no. 1, pp. 93–107, 1997.

[7] D. H. Johnson, “Point process models of single-neuron discharges,” *Journal of Computational Neuroscience*, vol. 3, no. 4, pp. 275–299, 1996.

[8] T. Ozaki, “Maximum likelihood estimation of hawkes’ self-exciting point processes,” *Annals of the Institute of Statistical Mathematics*, vol. 31, no. 1, pp. 145–155, 1979.

[9] Y. Ogata, “On Lewis’ simulation method for point processes,” *IEEE Transactions on Information Theory*, vol. 27, no. 1, pp. 23–31, 1981.

[10] K. Khanna, M. Smith, D. Wu, and T. Zhang, “Reconstructing the order book,” Stanford University, Tech. Rep., 2009.

[11] P. Hewlett, “Clustering of order arrivals, price impact and trade path optimisation,” in *Workshop on Financial Modeling with Jump processes, Ecole Polytechnique*, 2006.

[12] I. Allugundu, P. Puranik, Y. P. Lo, and A. Kumar, “Acceleration of distance-to-default with hardware-software codesign,” in *International Conference on Field Programmable Logic and Applications*, 2012, pp. 338–344.

[13] C. Guo, H. Fu, and W. Luk, “A fully-pipelined expectation - maximization engine for Gaussian mixture models,” in *International Conference on Field-Programmable Technology*, 2012, pp. 182–189.

[14] N. Weaver, “Retiming, repipelining, and c-slow retiming,” in *Reconfigurable Computing: the Theory and Practice of FPGA-based Computation*, S. Hauck and A. DeHon, Eds. Morgan Kaufmann, 2007.