Customisable Pipelined Engine for Intensity Evaluation in Multivariate Hawkes Point Processes

Ce Guo and Wayne Luk
Department of Computing
Imperial College London
ce.guo10,w.luk@imperial.ac.uk

Ekaterina Vinkovskaya
Department of Statistics
Columbia University
eov2101@columbia.edu

Rama Cont
Department of Mathematics
Imperial College London
r.cont@imperial.ac.uk

ABSTRACT

Hawkes processes are point processes that can be used to build probabilistic models to capture occurrence patterns of random events. They are widely used in high-frequency trading, seismic analysis and neuroscience. A critical calculation in Hawkes process models is intensity evaluation. The intensity of a point process represents the instantaneous rate of occurrence of events, but it is computationally expensive and challenging to calculate efficiently in order to make predictions using Hawkes process models. To accelerate the computation, we analyse data dependency in the intensity evaluation routine, and present a strategy to enable multiple intensity values to be computed with a single pass through the data. We then design and optimise a pipelined hardware engine based on our strategy. In our experiments, an FPGA-based implementation of the proposed engine is evaluated by four case studies. This implementation achieves up to 94 times speedup over an optimised CPU implementation with one core, and 12 times speedup over a CPU with eight cores.

1. INTRODUCTION

Hawkes processes [4] are point processes that can be used to build probabilistic models to capture occurrence patterns of random events. The study of Hawkes process models is attracting the attention of researchers and practitioners from various areas including high-frequency trading [1, 2], seismology [7, 9] and neuroscience [3, 6].

For instance, Hawkes processes have been widely used in financial modelling. They can generate accurate predictions of order flow [10] but have a heavy computational load. Since predictions are usually made at high frequencies and in real time, efficient implementations are key to success in practice.

A critical calculation in Hawkes process models is intensity evaluation. The intensity of a point process taken at a fixed point in time, t, is the instantaneous rate at time t with which events occur. The calculation of the intensity of the underlying Hawkes process is necessary in parameter estimation, simulation and prediction. The feature, however, that makes the Hawkes process useful for making predictions and modelling correlated data is that its intensity is stochastic, changes in time and depends on the history of all dimensions of the process. This makes the evaluation of the intensity a computationally demanding task.

For the purpose of obtaining predictions this evaluation must be performed as fast as possible. This efficiency problem has become increasingly serious in recent years because the length and dimension of real-world data sets have been growing swiftly. 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. The conflict between data size and computational efficiency is especially serious in time critical problems such as high-frequency trading and optimal order execution.

While data analysis systems can often 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 algorithms related to point processes. Our key contributions are as follows.

- We analyse the data dependence pattern of the intensity value computation. Using the result of the analysis, we develop a computing strategy where the intensity values of multiple dimensions can be computed with a single pass of the data.
- We identify the computationally intensive part of our strategy and propose a hardware architecture to handle this part. We also optimise the hardware by exploiting data parallelisation to make full use of the memory bandwidth.
- We evaluate an implementation of our proposed data flow engine with four case studies and various data sizes, and provide possible explanations to both expected and unexpected experimental observations.

The rest of the paper is organised as follows. Section 2 briefly describes point processes, Hawkes processes and the intensity evaluation problem. Section 3 presents an analysis of the data dependence pattern, and describes a strategy that can be used to compute the intensity values of multiple dimensions with a single pass through the data. Section 4 describes a hardware design that maps our strategy to a pipelined hardware engine, and a method to exploit data parallelisation. Section 5 provides experimental results about an implementation of our hardware design in four case studies, and explains experimental observations. Section 6 provides a brief conclusion, and describes possible future work.

This work was presented in part at the fourth international workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2013), Edinburgh, Scotland, United Kingdom, June 13-June 14, 2013.

2. BACKGROUND

In this section we provide a short introduction to Hawkes processes. We first illustrate related concepts including point processes and counting functions, and then briefly discuss the intensity evaluation problem.

2.1 Point Processes

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

The counting function $C(t)$ of a point process $\{t_i\}$ is defined by

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

In other words, the value of the counting function $C(t)$ is the number of elements in $\{t_i\}$ that are less than or equal to t. The *intensity function* $\lambda(t)$ of a point process $\{t_i\}$ is defined by

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

$$= \lim_{h \downarrow 0} \frac{1}{h} P(C(t+h) - C(t) > 0) \tag{3}$$

where $\mathbb{E}$ is the expectation operator; $P(A)$ is the probability of random event A. A *multivariate point process* with M dimensions, is in the form

$$\rho(M) = \big\{\{t_i\}_1, \{t_i\}_2, \dots, \{t_i\}_M\big\} \tag{4}$$

where $\{t_i\}_1, \{t_i\}_2, \dots, \{t_i\}_M$ are univariate point processes.

2.2 Hawkes Processes

A multivariate point process $\big\{\{t_i\}_1, \{t_i\}_2, \dots, \{t_i\}_M\big\}$ is a *multivariate Hawkes process* if for all dimensions $m \in 1..M$ the intensity function $\lambda_m(\cdot)$ of $\{t_i\}_m$ satisfies

$$\lambda_m(t) = \lambda_m^{\perp} + \sum_{m'=1}^{M} \int_0^t k(t-w) \mathrm{d}C_{m'}(w) \tag{5}$$

where $\lambda_m^{\perp}$ is a constant parameter; $C_{m'}(\cdot)$ is the counting function of $\{t_i\}_m$ and $k(\cdot)$ is a response function which needs to be integrable and positive. In this study, we forcus on the exponential response function defined by

$$k(u) = \alpha_{m,m'} e^{-u \cdot \beta_{m,m'}} \tag{6}$$

where $\alpha_{m,m'}$ and $\beta_{m,m'}$ are constant parameters. In other words, a parameter set for an M dimensional Hawkes process with an exponential response function is a triplet in the form

$$\mathcal{P} = \langle \lambda^{\perp}, A, B \rangle \tag{7}$$

where

$$\lambda^{\perp} = \begin{pmatrix} \lambda_1^{\perp} & \lambda_2^{\perp} & \dots & \lambda_M^{\perp} \end{pmatrix}^T \tag{8}$$

$$A = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,M} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{M,1} & \alpha_{M,2} & \cdots & \alpha_{M,M} \end{pmatrix} \tag{9}$$

$$B = \begin{pmatrix} \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,M} \\ \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{M,1} & \beta_{M,2} & \cdots & \beta_{M,M} \end{pmatrix} \tag{10}$$

where the decay parameters B can be reduced to an M dimensional vector by assuming that all rows are identical.

For example, a fragment of a two dimensional Hawkes process is shown in Figure 1. The crosses on the horizontal axes record the times of occurrences of the two event. The two serrated lines are the two intensity functions. Given a time point t, let $H(m,t)$ be the set of points that occur before t in the m-th dimension. In other words,

$$H(m,t) = \{t' : t' \in \{t_i\}_m \text{ and } t' < t\} \tag{11}$$

A *sample* or a *data set* of a Hawkes process up to time t, denoted by $D(t)$, is a set defined by

$$D(t) = \{H(1,t), H(2,t), \dots, H(M,t)\} \tag{12}$$

Intensity evaluation is the computation of the intensity values of all dimensions, $\lambda_1(t)$, $\lambda_2(t)$, $\dots$, $\lambda_M(t)$, given a parameter set $\mathcal{P}$, a time point t, and a data set $D(t)$.

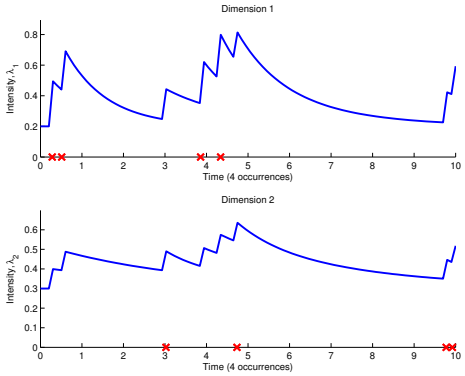


Figure 1: A Two Dimensional Hawkes Process

3. STRATEGY OF ACCELERATION

In this section, we first describe the data representation scheme adopted in this study, and then analyse the data dependence pattern in the intensity evaluation problem. We finally develop a computing strategy to calculate multiple intensity functions in parallel with a single data pass.

3.1 Data Representation

There are two major data representation schemes for point process data. One is to store the the points in all different dimensions in a single sequence. In this scheme, each point is recorded in the form (t_i, l_i) where t_i is the occurrence time and l_i is the corresponding dimension label.

The other data representation scheme is to store the points in different dimensions into separate sequences. More specifically, for an M dimensional point process, M sequences are created. Each sequence contains all points in a single dimension. With this scheme, dimension labels for the points are not required. In other words, the i-th event in the m-th dimension is denoted by its occurrence time $t_{m,i}$.

We selected the later data representation scheme because we consider it more suitable for hardware design than the former one. Not only does the absence of dimension label reduces the memory space requirement but it avoids redundant memory bandwidth consumption in data access. Furthermore, it brings simplicity for the hardware design and saves computing resources in the reconfigurable hardware.

3.2 Straightforward Computing Strategy

By Equation 5 and 6, the intensity function can be written in a computable form

$$\begin{aligned}\lambda_m(t) &= \lambda_m^{\perp} + \sum_{m'=1}^{M} \sum_{t' \in H(m',t)} k(t-t') \\ &= \lambda_m^{\perp} + \sum_{m'=1}^{M} \sum_{t' \in H(m',t)} \alpha_{m,m'} e^{-\beta_{m,m'} \cdot (t-t')} \end{aligned} \quad (13)$$

At a given time t, it is straightforward to compute the intensity values $\lambda_1(t)$, $\lambda_2(t)$, ..., $\lambda_M(t)$ individually using Equation 13. However, we consider such a strategy inefficient from the perspective of reconfigurable computing.

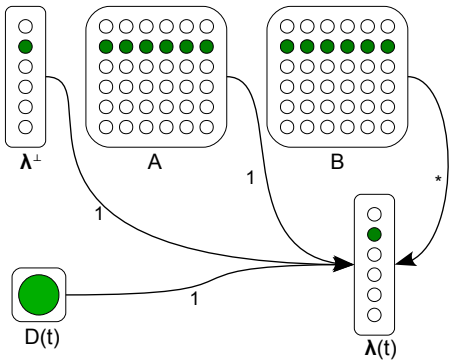


Figure 2: Data Dependence Pattern of the Original Intensity Evaluation Strategy

We can appreciate this efficiency problem by analysing the data dependency pattern which is shown in Figure 2. In this figure, an edge from a vertex x to vertex y describes the fact that x is a necessary element to compute y. If y can be computed with a single access to x, then the edge is labelled with '1', otherwise it is labelled with a star.

This data dependence pattern suggests that if we directly perform the computation following the definition, we may only obtain one intensive value after passing through the whole data set. Then it is unavoidable to access the whole data set for M times to compute all the M intensity values.

In this study we aim to design a hardware architecture to accelerate intensity evaluation. The bandwidth between the memory and computation unit is limited. Therefore we consider it memory-inefficient to obtaining only one intensity value with a data pass.

3.3 Computing Multiple Intensity Values with a Single Data Pass

With regard to the memory-efficiency problem discussed in the previous subsection, we study the possibility and methods of obtaining multiple intensity values simultaneously with a single data pass. Let $\lambda_{m_1}(t)$, $\lambda_{m_2}(t)$, ..., $\lambda_{m_c}(t)$ be a group of c intensity values at time t. By Equation 13,

$$\begin{pmatrix} \lambda_{m_1}(t) \\ \lambda_{m_2}(t) \\ \vdots \\ \lambda_{m_c}(t) \end{pmatrix} = \begin{pmatrix} \lambda_{m_1}^{\perp} \\ \lambda_{m_2}^{\perp} \\ \vdots \\ \lambda_{m_c}^{\perp} \end{pmatrix} + \mathbf{s}(t) \quad (14)$$

where

$$\mathbf{s}(t) = \sum_{m'=1}^{M} \begin{pmatrix} \sum_{t' \in H(m',t)} \alpha_{m_1,m'} e^{-\beta_{m_1,m'} \cdot (t-t')} \\ \sum_{t' \in H(m',t)} \alpha_{m_2,m'} e^{-\beta_{m_2,m'} \cdot (t-t')} \\ \vdots \\ \sum_{t' \in H(m',t)} \alpha_{m_c,m'} e^{-\beta_{m_c,m'} \cdot (t-t')} \end{pmatrix} \quad (15)$$

To compute these intensity values, one only needs to compute the M dimensional column vector $\mathbf{s}(t)$. We decompose this vector as

$$\mathbf{s}(t) = \sum_{m'=1}^{M} U_{m'} \mathbf{v}_{m'}(t) \quad (16)$$

where $U_{m'}$ is a $c \times c$ diagonal matrix defined by

$$U_{m'} = \begin{pmatrix} \alpha_{m_1,m'} & 0 & 0 & 0 \\ 0 & \alpha_{m_2,m'} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \alpha_{m_c,m'} \end{pmatrix} \quad (17)$$

and $\mathbf{v}_{m'}(t)$ is a c dimensional column vector defined by

$$\mathbf{v}_{m'}(t) = \sum_{t' \in H(m',t)} \begin{pmatrix} e^{-\beta_{m_1,m'} \cdot (t-t')} \\ e^{-\beta_{m_2,m'} \cdot (t-t')} \\ \vdots \\ e^{-\beta_{m_c,m'} \cdot (t-t')} \end{pmatrix} \quad (18)$$

The diagonal matrix $U_{m'}$ is irrelevant to the data. All the elements needed to construct $U_{m'}$ can be directly extracted from the m'-th column of the model parameter A. The vector $\mathbf{v}_{m'}(t)$ is the crucial part of the computation. Once $\mathbf{v}_{m'}(t)$ for all $m' \in 1..M$ are obtained, $\lambda_{m_1}(t)$, $\lambda_{m_2}(t)$, ..., $\lambda_{m_c}(t)$ can be computed effortlessly using Equation 14, 16, 17 and 18. However, the value of this vector depends on all historical events. Therefore the computation of this part becomes the efficiency bottleneck when the number of points in the data set is large.

Note that the data dependency underlying our proposed computation strategy shown in Figure 3 is different from that of the straightforward method shown in Figure 2. In particular, with our proposed strategy, the data set is only accessed once to compute all the c intensity values.

We consider this property valuable for developing hardware acceleration solution. In a hardware design, it is usually the memory access time, rather than the logical resources, that limits the performance and scalability of the hardware. Obtaining multiple results with one data pass

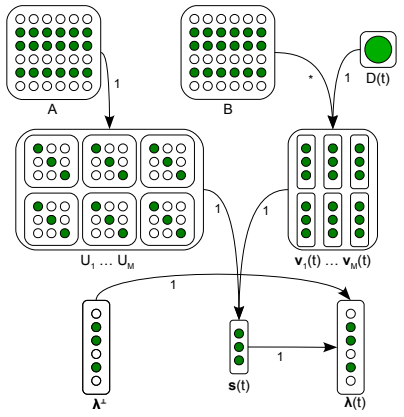


Figure 3: Data Dependence Pattern of our Proposed Evaluation Strategy

improves the memory efficiency. This is especially beneficial when the number of dimensions is large.

4. HARDWARE DESIGN

As we have discussed in the previous section, obtaining $\mathbf{v}_{m'}(\cdot)$ is the most computationally expensive process. In this section, we accelerate this part by designing and optimising a pipelined hardware architecture. We first propose a basic architecture and explain its core component, the *glove*. We then propose a method to combine multiple gloves to promote data parallelisation.

4.1 Basic Architecture

We propose an architecture, which is shown in Figure 4, to compute $\mathbf{v}_{m'}(\cdot)$ following Equation 18. The major business part of the architecture, which we call a *glove*, is enclosed in the large rounded square in the figure. For ease of discussion, we refer to a component in charge of accumulating the statistical information for a single dimension as a *finger*. It is necessary to deploy c fingers in a glove to compute the c intensity values in a single data pass.

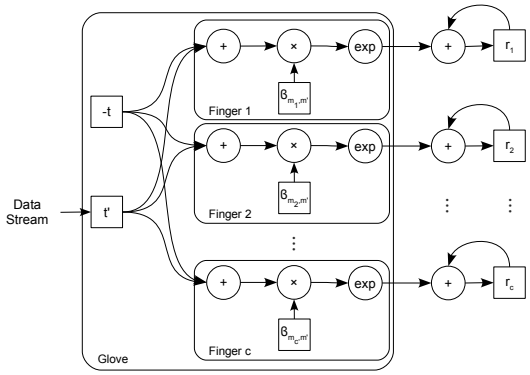


Figure 4: Basic Architecture with a Single Glove

The output of the i-th finger is accumulated using an adder and a chain of buffer registers, r_i. When a data instance in the m'-th dimension is streamed into the engine, the contribution of this data point towards all the c components in $\mathbf{v}_{m'}(t)$ computed and accumulated in parallel. When all the data instances in the m'-th dimension have been streamed into the engine, the entries in vector $\mathbf{v}_{m'}(t)$ can be harvested respectively from buffer chain r_1, r_2, ..., r_c. When all the data are processed, $\mathbf{v}_{m'}(t)$ for all $m' \in 1..M$ are obtained. The host computer then calculate $\lambda_{m_1}(t)$, $\lambda_{m_2}(t)$, ..., $\lambda_{m_c}(t)$ using Equation 14, 16, 17 and 18.

4.2 Data Parallelisation

An ideal situation for the single-glove architecture is that the number of fingers c is a factor of the number of dimensions M. In this situation, all the M intensity values can be computed with exact M/c data passes. No finger in the architecture is idle during the computation, and therefore the maximum performance can be achieved. If c is not a factor of M, we need $\lceil M/c \rceil$ data passes to finish the computation of all intensity values, and in some data passes, certain fingers of the glove are not active and therefore the maximum performance cannot be obtained. Practically, we may maximise c with respect to the limitation of hardware resources on a particular platform to allow the demonstration of maximum performance. However, we also expect to reduce c to avoid or reduce idleness.

Note that (i) the memory bandwidth requirement of the single-glove architecture is both small and constant; and (ii) there is no particular computational order in the summation operation in Equation 18. These two facts suggest that we may take advantage of data parallelism to maximise resource usage without increasing the number of fingers c.

We propose to deploy a group of g gloves in the way shown in Figure 5 to exploit the data parallelism. More specifically, we break data stream into g segments with identical or similar length and stream them into the g gloves in parallel. Then we add up the outputs of the i-th fingers of all gloves and accumulate the results.

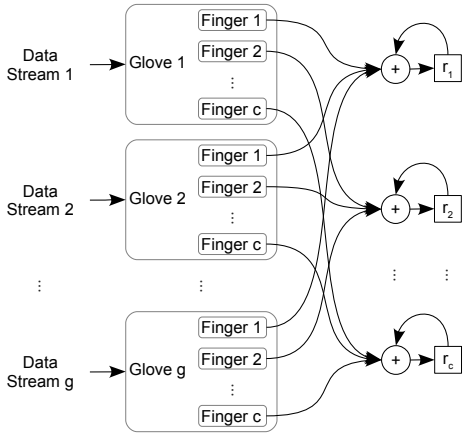


Figure 5: Exploiting Data Parallelisation by Combining g Gloves

5. EXPERIMENTAL EVALUATION

We implement our proposed architecture in an FPGA-based acceleration device and evaluate it by four case studies. In this section, we first present the general experimental settings and then present and discuss experimental results.

Table 1: Case Studies

No.	Area of study	M
1	Order Book Reconstruction (1 Stock) [10]	4
2	Earthquake analysis (10 Geological Areas) [9]	10
3	Stock Trading Behaviour Modelling (10 Stocks) [5]	20
4	Neurone Activity Modelling (32 Neurones) [3]	32

5.1 Experimental Settings

We implement our hardware design using a Maxeler MAX3 acceleration system in a fully-pipelined manner. This system is equipped with a Xilinx Virtex-6 FPGA. It communicates with the host computer via a PCI-Express interface. The hardware is described in the MaxJ language and compiled with Maxeler MaxCompiler.

The focus of this paper is to develop our parallelisation strategy and corresponding hardware design rather than exploiting the performance or resource efficiency of a particular implementation. In the experimental system, we deploy 4 gloves, each of which contains 32 fingers. The clock frequency of the FPGA is set to 100MHz for fast compilation.

We also build an implementation that runs exclusively on the CPU platform using the OpenMP library. To make a fair comparison, we perform a series of optimisations on this implementation, including avoiding redundant memory access and selecting best parallelisation scheme.

Both the host code for the FPGA implementation and the pure CPU implementation are written in the C programming language and compiled with the Intel C compiler with the highest compiling optimisation. The IEEE single precision floating point numbers are used in both hardware and software systems in our experiments. Both systems are invoked in a server with an Intel Xeon CPU running at 2.67GHz and 48GB DDR3 memory.

The general information of the case studies are listed in Table 1. For each problem, we generate test data sets with different sizes L from 10^4 to $10^{9.5}$, using the thinning procedure [8]. A data set is loaded in the main memory of the host computer before the computation begins. The data are then transmitted to the acceleration system via the PCI-Express interface in real time during the computation. The performance is measured by the execution time of computing all the M intensity values.

5.2 Results and Discussion

Experimental results are shown in Figure 6. Each column of the figures corresponds to a case listed in Table 1. The first two rows of figures record the performance of the FPGA and CPU implementations respectively. We record the data size in log scale with base 10 in these figures. To reflect the trend of the increment of the execution time, we also record the the execution time in log scale. Note that a small difference along the vertical axis in each figure means a huge difference in execution time due to the properties of the logarithm function. The third row of the figures shows the speedup of the FPGA implementation over one core and eight cores. We do not take the logarithms of these numbers.

The CPU implementation works well on eight cores. It is consistently around 7.5 times faster than a single core in all tested cases. The speedup of the FPGA implementation over the CPU implementation increases as the size of data set grows. The maximum speedup we observe across all the case studies is in Case 4 when the data size reaches $10^{9.5}$. The FPGA implementation is 94 times, and 12 times faster than the CPU implementation on a single core and eight cores respectively.

The execution times of the CPU implementation increase linearly as the data size grows. However, the growth pattern of the FPGA implementation appears irregular. In all cases except Case 1, there are huge leaps when the data size rises from 10^4 to 10^5. We are unsure about the reason behind these leaps, but they are likely to be caused by the hardware platform rather than the design. After this leap, the increment follows a convex function. Note that a convex increment pattern suggests a better performance than a linear one. It means that when the size of the data grows by h times, the execution time grows less than h times. This is because, while the execution time spent on the pipeline grows linearly with the size of the data set, the overhead, including hardware initialisation time and post-processing time of the host computer, does not increase.

The execution time for a fixed data size is similar across the four case studies regardless of the number of dimensions. This is not surprising because the numbers of dimensions in all cases are less than or equal to the number of fingers c in each glove. Therefore the engine is capable of handling a data set with one pass through the data. As a result, the time spent on the pipeline processing for difference data sets are identical, and the small difference in the total execution times is brought by the different efforts on post-processing times in the host computer.

6. CONCLUSION

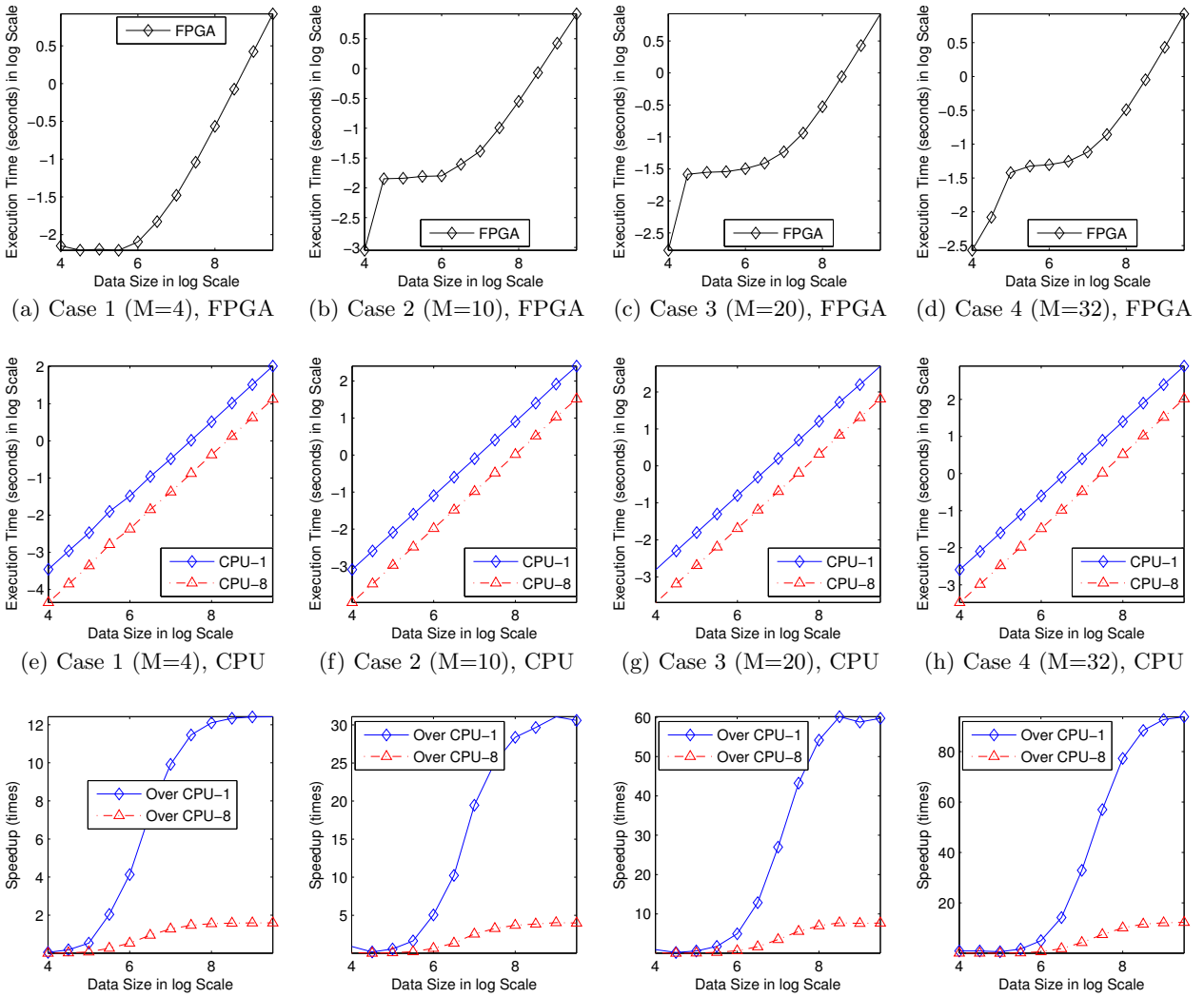
This paper presents a hardware architecture that accelerates intensity evaluation for multivariate Hawkes point processes. To the best of our knowledge, our work is the first to design hardware acceleration solution for point processes.

Rather than directly mapping the definition of the intensity values to a hardware architecture, we analyse the data dependence pattern in the computation, and present a strategy of acceleration to enable multiple intensity values to be computed with a single pass through the data. We then propose a hardware architecture based on our strategy, and optimise it by taking advantage of the data parallelism. Our experimental system implemented in a Vertex-6 FPGA achieves up to 94 times and 12 times speedup over a single-core CPU and an 8-core CPU respectively.

This work shows the potential of reconfigurable computing for accelerating point processes. Possible future work of this study includes (i) integrating the intensity evaluation engine with real-time predictive systems to forecast future occurrences of events in situations such as high-frequency trading; (ii) building performance models for the engine to predict the performance for different parameter settings or hardware platforms; (iii) developing acceleration systems for point processes on other acceleration systems such as graphics processing units (GPUs).

7. ACKNOWLEDGEMENTS

We thank the reviewers for their comments. 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.



(a) Case 1 (M=4), FPGA (b) Case 2 (M=10), FPGA (c) Case 3 (M=20), FPGA (d) Case 4 (M=32), FPGA

(e) Case 1 (M=4), CPU (f) Case 2 (M=10), CPU (g) Case 3 (M=20), CPU (h) Case 4 (M=32), CPU

(i) Case 1 (M=4), Speedup (j) Case 2 (M=10), Speedup (k) Case 3 (M=20), Speedup (l) Case 4 (M=32), Speedup

Figure 6: Experimental Results

8. REFERENCES

[1] E. Bacry, S. Delattre, M. Hoffmann, and J.F. Muzy. Modelling microstructure noise with mutually exciting point processes. *Quantitative Finance*, 13(1):65–77, 2013.

[2] C.G. Bowsher. Modelling security market events in continuous time: Intensity based, multivariate point process models. *Journal of Econometrics*, 141(2):876–912, 2007.

[3] R. Dahlhaus, M. Eichler, and J. Sandkühler. Identification of synaptic connections in neural ensembles by graphical models. *Journal of neuroscience methods*, 77(1):93–107, 1997.

[4] A.G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

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

[6] D.H. Johnson. Point process models of single-neuron discharges. *Journal of Computational Neuroscience*, 3(4):275–299, 1996.

[7] Y.Y. Kagan. Statistical distributions of earthquake numbers: consequence of branching process. *Geophysical Journal International*, 180(3):1313–1328, 2010.

[8] P.A. Lewis and G.S. Shedler. Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.

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

[10] E. Vinkovskaya. A point process model for the dynamics of limit order books. Submitted for publication, 2013.