Particle Filtering-based Maximum Likelihood Estimation for Financial Parameter Estimation

Jinzhe Yang
Imperial College & SWIP
London & Edinburgh, UK
jinzhe.yang@swip.com

Binghuan Lin
Techila Technologies Ltd
Tampere University of Technology
Tampere, Finland
binghuan.lin@techilatechnologies.com

Wayne Luk
Imperial College
London, UK
wl@doc.ic.ac.uk

Terence Nahar
SWIP
London, UK
terence.nahar@swip.com

***Abstract*—This paper presents a novel method for estimating parameters of financial models with jump diffusions. It is a Particle Filter based Maximum Likelihood Estimation process, which uses particle streams to enable efficient evaluation of constraints and weights. We also provide a CPU-FPGA collaborative design for parameter estimation of Stochastic Volatility with Correlated and Contemporaneous Jumps model as a case study. The result is evaluated by comparing with a CPU and a cloud computing platform. We show 14 times speed up for the FPGA design compared with the CPU, and similar speedup but better convergence compared with an alternative parallelisation scheme using Techila Middleware on a multi-CPU environment.**

I. Introduction

Particle Filter (PF), also known as Sequential Monte Carlo method (SMC), is a computationally intensive state estimation technique that is applied to solve dynamic problems involving non-normality and non-linearity. Information about how the method should evolve is contained in every single particle, and in order to acquire accurate estimation results, the number of particles is often large. Empirically, the more complex the system, the larger the amount of particles. A common concern is that the computational intensity for solving real-world problems using PF is high, and time constraint limits the applicability of PF to fields with runtime requirement. A detailed discussion of PF methods in practice can be found in [1].

Parameter estimation is one of the most crucial topics in the finance industry, especially for buy-side practitioners. The parameters are estimated from historical data and could be used to predict future movements of the market. Because of the lack of likelihood functions for jump diffusion models, practitioners use Bayesian methods as default estimation tools, such as Markov Chain Monte Carlo (MCMC) methods. This paper explores an alternative estimation method, the maximum likelihood estimation (MLE), for financial parameter estimation.

Some attempts have been made for parameter estimation using PF, but they only cover models without jump diffusion. The particles are treated as the evaluation function for the MLE, so the likelihood function could also be provided by the PF. The restriction of PF is that it brings heavy computational burden which might be infeasible for traditional platforms. It can take a few days or even up to weeks to calculate the result, which cannot meet runtime requirement in practice. In addition, high performance computing (HPC) technologies are still new to the finance industry. We provide a PF based MLE solution to estimate parameters of jump diffusion models, and we utilise HPC technology to speedup the estimation scheme so that it would be acceptable for practical use.

This paper presents the algorithm development, as well as the pipeline design solution in FPGA technology, of PF based MLE for parameter estimation of Stochastic Volatility with Correlated and Contemporaneous Jumps (SVCJ) [2]. The contributions of this paper are summarised as follows:

- We provide a PF-based MLE method that is parallelisable and robust. It can be a potential competitor for default estimation tools used for jump-diffusion models such as MCMC.
- We develop a collaborative CPU-FPGA platform for this method, making full use of both CPU and FPGA by resampling on multi-threaded CPU and enabling efficient evaluation of constraints and weights on FPGA.
- We offer optimisations through algorithmic method to minimise bottlenecks in complex computation, and though memory optimisation to reduce bandwidth overhead.
- We benchmark different parallelisation schemes of particle filtering on different computing platforms.

II. Background

Introducing jump components of diffusion models used in finance is one of the main developments in financial modeling literature during the past decade. This paper focuses on the affine jump diffusion model, which has good analytical properties. On the one hand, this method allows the derivation of semi-closed form for option pricing formulae that enable fast option pricing. This feature is a necessary criterion of the model to be acceptable by the financial industry. On the other hand, this model captures large movements in the financial market by incorporating jump components.

In our approach, the joint dynamic (SVCJ) of log-price s_t and stochastic variance V_t is given by:

$$ds_t = (\mu - 1/2V_t)dt + \sqrt{V_t}dW_t^s + d(\sum_{j=1}^{N_t} Z_j^s) \qquad (1)$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t}dW_t^v + d(\sum_{j=1}^{N_t} Z_j^v) \qquad (2)$$

where $N_t \sim poi(\lambda dt)$ is a Poisson distributed random number that describes the number of jumps, $Z_j^s = \mu_s + \rho_s Z_j^v + \sigma_s \epsilon_j$, $\epsilon_j \sim N(0,1)$ (normal distribution) and $Z_j^v \sim exp(\mu_v)$ (exponential distribution) are random numbers that describe the jump size in log-price process and variance process, respectively.

The set of parameters from the SVCJ model that we estimate are: μ, μ_v, μ_y, κ, θ, σ_v, σ_y, ρ, ρ_j and λ, where

- μ: long term mean of the return
- μ_v: rate of exponential jump size in volatility
- μ_y: mean of jump size in return
- κ: rate of mean reversion
- θ: long term variance
- σ_v: volatility of volatility
- σ_y: volatility of jump size in return
- ρ: correlation between volatility process and return process
- ρ_j: correlation between jumps in return and in volatility
- λ: jump intensity

Some of the parameters above are not provided in formula 1 and 2, because they appear when the model is discretised: it is transformed into functions of finite set of state variables with known joint distribution. It is a generic model representing jump-diffusion models:

- When $\lambda = 0$ (SV), it is a Heston Model [3].
- When $\mu_v = 0$ (SVJ), it is a Bates Model [4].

In other words, the model we estimate is more general than several widely used financial models. Following the literature, we assume two Brownian motions W^s and W^v are correlated with correlation coefficient ρ. This enables the model to capture the well-known leverage effect in financial time series.

III. Algorithm development

Estimation of diffusion models in finance has long been a challenge due to the existence of latent states and the discreteness of observable data. Mixing a jump process and the original process naturally introduces more difficulties by providing more flexibility to the model.

Maximum likelihood estimation (MLE) is a widely used statistical tool to estimate the parameters of a model. By maximising the likelihood function, the estimation procedure fits the model to the observed data. We use stock prices as a set of the observations and variance as the latent states.

We first develop a Sequential Importance Resampling (SIR) strategy for our parameter estimation. It is a standard strategy for models with/without jumps. However, the result accuracy are different between the two kinds of models. For models without jumps, such as the Heston Model, it could provide accurate parameter sets. Nevertheless, when it is applied to jump-diffusion models, where we take distribution $q(V_{t+1}|V_t^i, s_{t+1})$ as prior distribution $p(V_{t+1}|V_t^i)$, we may suffer from sample impoverishment of the particles in the occurrence of jumps [5].

We extend the SIR algorithm to become the auxiliary particle filter (APF) to handle such problem. In contrast to a standard SIR particle filter, where the sampling of V_{t+1} is blind of s_{t+1}, APF samples V_{t+1} from $p(V_{t+1}|V_t, s_{t+1})$, and using auxiliary variables as an approximation when the exact distribution of $p(V_{t+1}|V_t, s_t + 1)$ is not available.

Based on [6] and [7], we extend the algorithm to the SVCJ model.

Algorithm 1 APF

```
for t=1 to T do
    for n=1 to N do
        Generate auxiliary variable of volatility as the expected integrated volatility: $\hat{v}_t = V_{t-1} + \kappa(\theta - V_{t-1}) + \rho\sigma_v(y_{t-1} - \mu + \frac{1}{2}V_{t-1} - J_{t-1}) + J_{t-1}\mu_v$
        Compute first stage weights $w_t \propto p(s_t|s_{t-1}, \hat{v}_t)$
    Resample $V_{t-1}$, $\hat{v}$ according to the first stage weights.
    for n=1 to N do
        Propagating the current particles by generating:
        Number of jumps $J$,
        Jump sizes in return $Z_y$
        Jump sizes in volatility $Z_v$ using $\hat{v}_t$;
        Generating volatility from $p(V_t|s_t, s_{t-1}, J, Z_v, Z_y)$
        Calculate second stage reweighting $\pi$ as in standard auxiliary particle filter using importance sampling rule.
    Resample $V$, $J$, $Z_v$, $Z_y$
```

According to [8], the likelihood function can be derived as:

$$LIK = \hat{p}(s_1)\prod_{t=2}^{T}\hat{p}(s_t|s_{t-1}) \qquad (3)$$

As proved in [9], the likelihood estimator $\mathbf{L_N^d}$ is an unbiased estimator of the likelihood $\mathbf{L^d}$ of the state space model (Euler discretisation of the original jump-diffusion process).

Furthermore, the likelihood of the discrete process converges to the likelihood of the diffusion process. Denote the set of model parameters by θ. As $\Delta t \to 0$,

$$\mathbf{L^d}(\theta) \to \mathbf{L^c}(\theta) \qquad (4)$$

where $\mathbf{L^c}$ is the likelihood of the original jump diffusion process.

Since we can get an estimation of likelihood as a byproduct of the particle filtering procedure, it is straightforward to construct the particle filtering-based MLE by taking the estimator based on particle filtering as the objective function of an optimisation algorithm. To summarise the MLE procedure: for each iteration of MLE, we input a set of estimated parameters, the PF outputs all corresponding results and evaluates the likelihood function, the optimiser refers to the likelihood and generates another set of parameters. The iteration terminates when the likelihood converges or reaches the maximum iteration. The resulting parameter set is the one with the highest likelihood.

IV. Hardware Design

A. System

The computational cost of the MLE procedure depends on the optimiser and evaluation of the likelihood function. If the evaluation of the likelihood function is cheap and the optimiser is fast, then the MLE is cheap. However, this is not true for jump-diffusion models. First, the likelihood estimator approximated by particle filter is non-smooth with respect to model parameters, disabling the use of gradient-based optimisers. Second, to have a good approximation of the likelihood, a large number of particles could be needed. As a result, the evaluation of the likelihood function is very time consuming.

The propagation of particles is parallel; exchange of information (data) is only necessary during the stage of normalising weight and resampling.

Due to the need for resampling and normalisation, accelerating our scheme has two main challenges:

- Heavy calculations inside the particles.
- Large amount of particles for resampling.

The difficulty behind each challenge is: computing resource and random memory access, respectively. FPGA data-flow paradigm performs ideally only when data are accessed sequentially. Otherwise, much of the bandwidth is wasted. Especially when sampling from very large amount of numbers, the random memory access would be the performance bottleneck, and to our knowledge, when the population is as large as our case, sampling method based on FPGA could be much slower than the CPU. On the other hand, the CPU has multiple levels of cache that enables fast and flexible memory access. Since resampling would take twice in a single time step, we design our system to take advantage of both CPU and FPGA: on FPGA, we exploit spatial parallelism and process the particles in a pipelined data path, and on the CPU, the flexible and fast memory allocation could provide significantly better performance for resampling. In addition, a resampling process with very small population would also be processed on FPGA.

The system design for our approach can be found in Figure 1. Kernel 1 calculates the auxiliary variable of volatility as the expected integrated volatility, which is also the calculation inside the first loop in algorithm 1. The CPU concurrently generates the random numbers which could be used for the resampling and kernel 2. After CPU receives all the expected volatility and weights from kernel 1, it starts resampling based on the weights, and the resampled data, as well as the random numbers, would be the input to kernel 2. Kernel 2 handles the second stage reweighting, which is the second loop in algorithm 1. The CPU and the FPGA collaborate similarly on the resampling process after kernel 1.

B. Kernel

As for the pipelined PF design inside the kernel, according to the formulae in the previous section, there are three random numbers for a single time step: two normally distributed

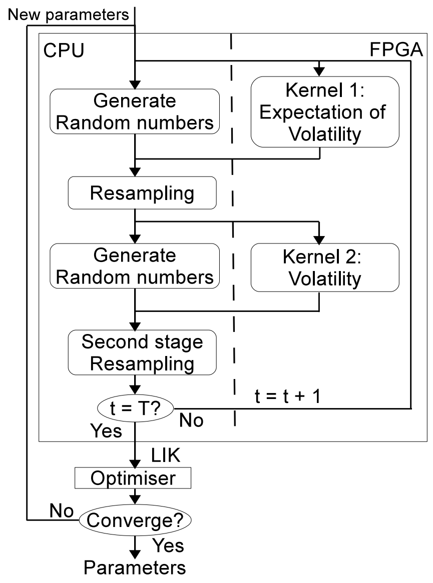

Fig. 1. Overall Design

random numbers for describing the movement of s and V, and an exponentially distributed random number for ϵ_v. Three challenges are identified:

1) Heavy data transfer requirement between CPU and FPGA for each iteration.
2) Calculation for normal probability density function and Poisson probability density function requires many resources as they are both frequently used.
3) When calculating the weights, a random sample process with small population is involved, and transferring the data back to CPU for the resampling is significantly inefficient.

To address challenge 1, we provide a scheme that reduces I/O streams for data transfer between iterations. Regarding four random numbers are needed for each particle at each iteration (the additional one is a uniform distributed random number between [0,1], for the resampling process with small population on FPGA), and the random numbers are independent with the resampling process, the CPU generates the random numbers for the next iteration while the FPGA is resampling for the current iteration. When the FPGA finishes transferring variable streams to the CPU, the CPU writes the generated random numbers to DRAM on the FPGA card concurrently with the resampling process in the CPU. The random numbers could be directly read from DRAM by the kernel.

Challenge 2 implies avoiding kernel divide operators and control for factorial. We avoid using divide operators in two ways, attempts, adjusting the calculation sequence to merge different divide operators, and pre-calculating constant dividends on the CPU and sending the FPGA their reciprocal. Also, we pre-calculate the reciprocal of factorial results that could be used for Poisson probability distribution function. Those pre-calculated results are uploaded to FPGA memory initially.

To address challenge 3, we formalise the weights for resampling, apply an accumulator for adding the weights, and a counter for recording the index. For each step, we sequentially add a weight to the accumulator, and the counter adds one. After comparing the number inside the accumulator with the random number, we output the result in the accumulator as well as the counter, if the number inside the accumulator is greater than the random number; otherwise we continue the step by accepting the next weight.

V. RESULT

Our FPGA-based data pipelines have the bottleneck of data size in both CPU and FPGA. For CPU, to achieve high accuracy, the number of particles is set to be 1 million. For the FPGA pipeline, the high cost of I/O bandwidth restricts our performance: the bandwidth is almost fully occupied while the hardware usage is only 55%. Thus, disregarding the bandwidth bottleneck, our performance should be nearly doubled. To make a fair comparison, our single-core sequential code and the host code for FPGA are written in C and compiled by the Intel C Compiler. The CPU we use both for software execution and for FPGA host is a 2.67GHz Xeon 5650 multicore processor, with 12 cores and 24 threads. Our FPGA result is based on Vertex-6 SX4757, with a clock frequency of 50MHz.

For the cloud computing version, we apply an external scheme. We divide the particle swarm of N particles into M groups, each of which containing K particles, where $N = M \times K$, and the likelihood estimator is given by:

$$\mathbf{L}^{\mathbf{d}}_{\mathbf{N}}(\theta) \approx \frac{1}{M}\mathbf{L}^{\mathbf{d}}_{\mathbf{K}}(\theta) \tag{5}$$

It is called an external scheme since in our FPGA design, resampling process would take place for all particles; however, our cloud implementation uses local resampling scheme within each node. More precisely, when particle shrinks at a node, resampling is processed only for the particles in this node [10]. The external scheme is simple and requires less implementation effort. We take $M = 20$, so we can maintain a million particles on 20 nodes. The acceleration factor is approximately 14 times. The required quantity of K is expected to increase with the frequency of jumps. However, the disadvantage of local resampling (for the external scheme) is its worse convergence than the internal scheme performed by our PFGA design.

	CPU	CPU+FPGA	speedup
1st Stage PF	35.54s	8.07s	4.4x
2nd Stage PF	159.93s	8.09s	19.8x
Total PF	195.47s	16.16s	12.1x
Total	408.47s	24.53	16.7x

TABLE I
CPU AND FPGA PERFORMANCE, 1 MILLION PARTICLES

	LUTs	FFs	BRAMs	DSPs
Total Available	297600	595200	1064	2016
Total Resource Used	234635	342461	237	1124
% Resource Used	78.84%	57.54%	22.27%	55.75%

TABLE II
FPGA USAGE INFORMATION

Table I summarises the performance of CPU and reconfigurable system, while Table II provides the hardware resource usage information. Table III compares the pros and cons of the FPGA and the Cloud implementations.

	FPGA	Cloud
Parallel Scheme	Internal	External
Resampling	Overall Resampling	Local Resampling
Convergence	Better	Worse
Development	Costly	Easy
Power Consuption	Energy Efficient	Power Consuming

TABLE III
COMPARISON BETWEEN FPGA & CLOUD

VI. CONCLUSION

This paper has demonstrated how PF is accelerated using FPGA technology, which provides a promising solution for parameter estimation of SVCJ. The FPGA design is shown to be faster and more energy efficient than those based on CPU and Cloud. The estimated parameters could be used for future usage such as pricing a financial instrument.

We are working on two improvements: clock frequency and the resampling scheme. Currently the clock frequency is 50MHz, because we employ floating-point arithmetic for our FPGA design. By adopting fixed-point arithmetic, the clock frequency could rise easily. Another improvement is to merge the two resampling steps while guaranteeing accuracy, because resampling twice is the most time consuming part so far.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n 289032. This work is also supported in part by the UK EPSRC, by the Maxeler University Programme, by the HiPEAC NoE, by Altera, and by Xilinx.

REFERENCES

[1] A. Doucet, N. d. Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice*. Springer, 2001.

[2] D. Duffie, J. Pan, and K. Singleton, “Transform analysis and asset pricing for affine jump-diffusions,” *Econometrica*, vol. 68, no. 6, pp. 1343–1376, 2000.

[3] S. L. Heston, “A closed-form solution for options with stochastic volatility with applications to bond and currency options,” *Review of financial studies*, vol. 6, no. 2, pp. 327–343, 1993.

[4] D. S. Bates, “Post-’87 crash fears in the s&p 500 futures option market,” *Journal of Econometrics*, vol. 94, no. 1-2, pp. 181–238, 2000.

[5] B. Lin, “State filtering of jump diffusion model,” 2013.

[6] M. K. Pitt and N. Shephard, “Filtering via simulation: Auxiliary particle filters,” *Journal of the American statistical association*, vol. 94, no. 446, pp. 590–599, 1999.

[7] M. S. Johannes, N. G. Polson, and J. R. Stroud, “Optimal filtering of jump diffusions: Extracting latent states from asset prices,” *Review of Financial Studies*, vol. 22, no. 7, pp. 2759–2799, 2009.

[8] M. K. Pitt, “Smooth particle filters for likelihood evaluation and maximisation,” *Warwick Economic Research Papers*, 2002.

[9] P. Del Moral, *Feynman-Kac formulae: genealogical and interacting particle systems with applications*. Series: Probability & Applications Springer Verlag, 2004.

[10] J. Míguez, “Analysis of parallelizable resampling algorithms for particle filtering,” in *Signal Processing*, 2007, pp. 3155–3174.