A heterogeneous computing approach to maximum likelihood parameter estimation for the Heston model of stochastic volatility

A. S. Hurn[1] K. A. Lindsay[2] D. J. Warne[3]

(Received 4 January 2016; revised 21 November 2016)

Abstract

Stochastic volatility models are of fundamental importance to the pricing of derivatives. One of the most commonly used models of stochastic volatility is the Heston model in which the price and volatility of an asset evolve as a pair of coupled stochastic differential equations. The computation of asset prices and volatilities involves the simulation of many sample trajectories with conditioning. The problem is treated using the method of particle filtering. While the simulation of a shower of particles is computationally expensive, each particle behaves independently making such simulations ideal for massively parallel heterogeneous computing platforms. We present a portable OpenCL implementation of the Heston model and discuss its performance and

DOI:10.21914/anziamj.v57i0.10425, © Austral. Mathematical Soc. 2016. Published November 28, 2016, as part of the Proceedings of the 12th Biennial Engineering Mathematics and Applications Conference. ISSN 1445-8810. (Print two pages per sheet of paper.) Copies of this article must not be made otherwise available on the internet; instead link directly to the DOI for this article. Record comments on this article via
http://journal.austms.org.au/ojs/index.php/ANZIAMJ/comment/add/10425/0

efficiency characteristics on a range of architectures including Intel CPUs, Nvidia GPUs, and Intel Many-Integrated-Core accelerators.

Contents

1 Introduction

Stochastic volatility models are fundamental tools in the pricing of derivative contracts such as European options. However, the difficulty is that these models rarely have closed-form transitional density functions, and consequently their practical application is normally a computationally intensive task [1, 5].

Hurn et al. [5] recently proposed that graphics processing units (GPUs) be used to improve the performance of parameter estimation for financial models using index data, including options written on that index. We propose a more general heterogeneous computing solution which exploits parallelism in many different hardware architectures. The Nelder–Mead algorithm [9] is used within a maximum likelihood framework to estimate the parameters of the Heston stochastic volatility model [4] from index and option data on the S&P 500 index between 1st January, 1990 and 30th June, 2012.

The primary contribution of this work is the computational analysis of the particle filtering method used by Hurn et al. [5, 6] in a general heterogeneous computing context. Our findings suggest that the difference in performance benefit from GPUs over other architectures may not be sufficiently significant to warrant development of codes that solely target GPUs.

1.1 The Heston model

Given independent Wiener processes $W_1(t)$ and $W_2(t)$, the Heston stochastic volatility model with respect to the physical measure is given by the stochastic differential equations (SDEs),

$$\frac{dS}{S} = (r - q - \xi_S V)\,dt + \sqrt{V}\left(\sqrt{1-\rho^2}\,dW_1 + \rho\,dW_2\right), \tag{1}$$

$$dV = \kappa_P\,(\gamma_P - V)\,dt + \sigma\sqrt{V}\,dW_2\,, \tag{2}$$

where $S(t)$ and $V(t)$ are the index and volatility processes respectively, r is the risk-free rate of interest, q is the dividend-price ratio, ξ_S is the equity premium, γ_P is the long-time mean volatility, κ_P is the rate at which $V(t)$ reverts to γ_P, σ is the volatility of volatility, and ρ is the correlation between returns and volatility. The role of the equity premium in the physical model (equations (1) and (2)) is to compensate a risk adverse investor for exposure to equity risk.

Options are priced under the risk neutral measure

$$\frac{dS}{S} = (r-q)\,dt + \sqrt{V}\left(\sqrt{1-\rho^2}\,dW_1 + \rho\,dW_2\right), \tag{3}$$

$$dV = \kappa_Q\,(\gamma_Q - V)\,dt + \sigma\sqrt{V}\,dW_2\,, \tag{4}$$

where κ_Q and γ_Q are related to the parameters of the physical model (equations (1) and (2)) by the formulae $\kappa_Q = \kappa_P + \lambda\sigma^2$ and $\gamma_Q = \kappa_P\gamma_P/\kappa_Q$. The task is to estimate the values of the parameters $\theta = \{\rho, \kappa_P, \xi_S, \sigma, \kappa_Q, \gamma_Q\}$ from index and option data on the S&P 500, a task that necessarily involves both the physical (1)–(2) and risk neutral model (3)–(4).

2 Parameter estimation

Consider a system observed at discrete time points $t_0, t_1, \ldots, t_T$, with X_i the observation at time t_i and T is the number of observation times (excluding the initial condition). Given a likelihood function $\mathcal{L}(\theta; X_0, \ldots, X_T)$, classical parameter estimation computes the maximum likelihood estimator (MLE) for the parameter set θ responsible for the observations:

$$\theta_{\text{MLE}} = \arg\max_{\theta} \hat{l}\,(\theta; X_0, \ldots, X_T)\,, \tag{5}$$

where

$$\hat{l}\,(\theta; X_0, \ldots, X_T) = \frac{1}{T}\log\mathcal{L}\,(\theta; X_0, \ldots, X_T)\,. \tag{6}$$

In the case of the stochastic volatility model (equations (1)–(4)) the observations have the form $X_i = \left(S_i, V_i, H_i^{(1)}, \ldots, H_i^{(M)}\right)$, where S_i, V_i, $H_i^{(j)}$ and M are respectively the index, volatility, the price of the jth option on the index at time t_i, and the number of options available on the index. The average log-likelihood for this problem is

$$\hat{l}\left(\theta; X_0, \ldots, X_T\right) = \frac{1}{T}\left[\sum_{j=1}^{M} \log g\left(H_0^{(j)} \mid \tilde{H}_0^{(j)}; \theta\right)\right.$$
$$\left. + \sum_{i=1}^{T}\left(\log f_P\left(X_i, t_i - t_{i-1} \mid X_{i-1}; \theta\right) + \sum_{j=1}^{M} \log g\left(H_i^{(j)} \mid \tilde{H}_i^{(j)}; \theta\right)\right)\right], \quad (7)$$

where f_P is the transitional density function for the physical model (1)–(2), $\tilde{H}_i^{(j)}$ is the predicted option price under the risk neutral model (3)–(4) for the jth option at time t_i, and g is a known distribution of option pricing errors [2, 5].

The evaluation of expression (7) presents two challenges. First, it contains volatility which is an unobservable variable, and second, the evaluation of g requires the calculation of many model option prices under the risk-neutral measure.

2.1 Recursive filtering

Volatility is a latent variable of the problem, nevertheless information about unobserved volatility is inferred from historical observations $Z_i = \{X_k\}_{k=0}^{k=i}$. The calculation of total likelihood (equation (7)) proceeds incrementally as the state and volatility are advanced from time t_{i-1} to t_i using recursive particle filtering [5, 7].

Assume that $f\left(V_{i-1} \mid Z_{i-1}\right)$ is a known filtered probability density function (PDF) for volatility given historical data up to t_{i-1}. Bayes' Theorem provides the equivalent PDF for t_i, namely

$$f\left(V_i \mid Z_i\right) = \frac{f\left(X_i, V_i \mid Z_{i-1}\right)}{f\left(X_i \mid Z_{i-1}\right)}. \quad (8)$$

The right hand side of equation (8) is evaluated using the integrals

$$f(X_i, V_i \mid Z_{i-1}) = \int_{\mathcal{V}} f(X_i \mid V_i, V_{i-1})\, f(V_i \mid V_{i-1})\, f(V_{i-1} \mid Z_{i-1})\, dV_{i-1}\,, \quad (9)$$

$$f(X_i \mid Z_{i-1}) = \int_{\mathcal{V}} f(X_i, V_i \mid Z_{i-1})\, dV_i\,, \quad (10)$$

where $\mathcal{V}$ is the state space of volatility. The integrals in equations (9) and (10) are evaluated numerically using Monte Carlo methods. The most computationally intensive component of this process is the evaluation of the function $f(X_i \mid V_i, V_{i-1})$ in equation (9) as it requires the evaluation of option prices conditioned on V_i at t_i.

2.2 Computation of option prices

Given a call option with strike price K, maturity T and index spot price S_0, the expected payoff is

$$\tilde{H} = \int_K^\infty (S - K) \int_{-\infty}^\infty f_Q(S_0, V, T, \mid S, v; \theta)\, dS\, dv\,. \quad (11)$$

Here f_Q is the transitional density function for the risk neutral measure. For brevity, we restrict our attention to the pricing of a call option, but the price of a put option is calculated in a similar way.

The crucial idea is that the characteristic function of the risk neutral model (equations (3) and (4)) is computed in semi-closed form. The integral contained within the payoff function is then approximated accurately from this semi-closed expression.

The calculation for the Heston model proceeds as follows. Let $Y = \log S/S_0$ and substitute into equation (3) using Itō's Lemma to obtain

$$dY = \left(r - q - \frac{V}{2}\right) dt + \sqrt{V}\left(\sqrt{1-\rho^2}\, dW_1 + \rho\, dW_2\right). \quad (12)$$

The expected payoff for the call option in terms of Y is

$$\tilde{H} = S_0 \int_{\log \xi}^{\infty} (e^y - \xi) \int_{-\infty}^{\infty} f_Q\,(0, V, T, \mid y, v; \theta)\, dy\, dv\,, \tag{13}$$

where $\xi = K/S_0$.

The backward Kolmogorov equation describing the evolution of the transitional density function of Heston's risk neutral model with respect to the initial state satisfies the partial differential equation

$$\begin{aligned} \frac{\partial f_Q}{\partial t} = &-\left(r - q - \frac{V}{2}\right) \frac{\partial f_Q}{\partial Y} - \kappa_Q\,(\gamma_Q - V)\,\frac{\partial f_Q}{\partial V} \\ &- \frac{V}{2}\left[\frac{\partial^2 f_Q}{\partial Y^2} + 2\rho \frac{\partial^2 f_Q}{\partial Y \partial V} + \sigma^2 \frac{\partial^2 f_Q}{\partial V^2}\right]. \end{aligned} \tag{14}$$

The Fourier transform of equation (14) is now taken to obtain an equation satisfied by the characteristic function of f_Q, namely

$$F_Q\,(Y, V, t, \omega_y, \omega_v; \theta) = \iint_{\mathbb{R}^2} f_Q\,(Y, V, t \mid y, v; \theta)\, e^{i(\omega_y y + \omega_v v)}\, dy\, dv\,, \tag{15}$$

which is a function of frequencies ω_y and ω_v. The equation satisfied by expression (15) is then observed to have a semi-closed form solution given by the anzatz

$$F_Q\,(Y, V, t, \omega_y, \omega_v) = e^{B_0(\tau,\omega_y,\omega_v) + B_1(\tau,\omega_y,\omega_v)Y + B_2(\tau,\omega_y,\omega_v)V}, \tag{16}$$

with $\tau = T - t$, and in which the functions B_0, B_1 and B_2 satisfy a set of ordinary differential equations (ODEs) that must be solved numerically. These ODEs are obtained via the Fourier transform of equation (14); Hurn et al. [5] described details.

Typically f_Q is well approximated by a function of compact support over $y \in [-\beta, \beta]$ for sufficiently large values of β. In this case the integral of f_Q

with respect to ν is represented with high accuracy by the Fourier series [6],

$$\int_{-\infty}^{\infty} f_Q\left(Y, V, T \mid y, \nu; \theta\right) d\nu = \sum_{k=-\infty}^{\infty} c_k e^{-k\pi i y/\beta},$$

with coefficients determined from the solution of equation (16), namely

$$c_k \approx \frac{1}{2\beta} F_Q\left(Y, V, T, \omega_k, 0; \theta\right), \quad \omega_k = \frac{k\pi}{\beta}.$$

This Fourier series approximation method is applicable to a wider class of options pricing models beyond the Heston model. Hurn et al. [6] provided more detailed discussion and analysis.

The successful implementation of the above steps allows the calculation of expression (13) that, in turn, completes the Monte Carlo integration for equation (9).

Algorithm 1 details the evaluation of the log-likelihood function (equation (7)). The constants N_t, N_b, N_p, N_s, and N_f are, respectively, the number of observations, burn-in simulations, particles, Heston simulations and Fourier frequencies.

3 Heterogeneous computing implementation

In this section we develop the main components of a portable heterogeneous implementation to the MLE developed in Section 2. The Open Computing Language (OpenCL) programming architecture is presented along with the advantages which make it a viable alternative to vendor specific languages such as the CUDA C language provided by Nvidia.

Algorithm 1 Particle filter for log-likelihood function, $\hat{l}(\theta; X_0, \ldots, X_T)$ evaluation.

```
1: Input $\theta = \{\rho, \kappa_P, \xi_S, \sigma, \kappa_Q, \gamma_Q\}$ and $X_i = \{S_i, H_i^{(1)}, \ldots, H_i^{(M)}\}$.
2: for $k = 1, \ldots, N_f$ do
3:     Compute $F_Q$ coefficients $B_0(\tau, \omega_k, 0)$, $B_1(\tau, \omega_k, 0)$ and $B_2(\tau, \omega_k, 0)$.
4: end for
5: Initialise particles $\{w_0^k, V_0^k\}$ using burn-in simulations.
6: for $i = 1, 2, \ldots, N_t$ do
7:     for $k = 1, 2, \ldots, N_p$ do
8:         Simulate physical model forward to obtain $V_i^k \sim f(V_i \mid V_{i-1})$ .
9:         Compute option prices $\tilde{H}_i^{(j)}$ using Fourier series approximation.
10:        $w_i^k \leftarrow \log f\left(S_i, \tilde{H}_i^{(1)}, \ldots \tilde{H}_i^{(M)} \mid V_i^k, V_{i-1}^k\right) + \sum_{j=1}^{M} \log g\left(H_i^{(j)} \mid \tilde{H}_i^{(j)}\right)$ .
11:    end for
12:    Importance re-sampling of particles $\{w_i^k, V_i^k\} \sim f(V_i \mid Z_i)$ .
13: end for
14: Accumulate log-likelihood, $\hat{l} \leftarrow 1/N_t \sum_{i=0}^{N_t} \sum_{k=1}^{N_p} w_i^k$ .
15: return $\hat{l}$
```

3.1 The Open Computing Language

OpenCL is an open standard for heterogeneous computing developed and maintained by the Khronos Group [3]. The standard provides a general framework for developing highly parallel algorithms. Defined in the standard is a parallel programming language, a parallel device execution model and a C application programming interface library for serial programs to manage and utilises parallel devices.

The OpenCL standard abstracts any device capable of parallel code execution. A device is comprised of asynchronous compute units, each of which is composed of processing elements that execute in a single instruction, multiple data fashion. Parallel functions in OpenCL, called *kernels*, define the operations of a *work item* which executes on a processing element. Work items can be grouped to ensure that they execute on the same compute unit and share local memory.

One of the main advantages of OpenCL is that any device that has a vendor supported implementation of OpenCL can be targeted. Furthermore, the target need not be selected until runtime, allowing the code to query available vendors and devices before compilation of device programs. The second important advantage of OpenCL is that it automatically enables reasonably fair benchmark comparisons of architectures. This is an important point and the focus of this article as the common rhetoric of GPUs giving $100\times$ speed improvements is often due to poor device comparisons [8].

3.2 Parallel particle filter

Two components of our algorithm lend themselves to parallelisation. First, a shower of particles is used for the Monte Carlo integration of equations (9) and (10). Each forward step in the calculation of the likelihood function is parallelised as particles execute independently of each other. Second is the evaluation of the coefficients of the approximating Fourier series of the transitional density.

Given N particles sampled from $f(V_{i-1} \mid Z_{i-1})$, the following actions are performed for each particle.

- Simulate the Euler–Maruyama discretisation of the physical model forward to t_i to sample $f(V_i \mid V_{i-1})$. This is given in line 8 of Algorithm 1.
- Evaluate $f(X_i \mid V_i, V_{i-1})$ as provided in Section 2. This includes evaluation of Fourier coefficients and option prices (lines 3 and 9 of Algorithm 1).

The results of the particle shower are then accumulated to generate the final likelihood contribution for that time period. A new set of particles are then generated from $f(V_i \mid Z_i)$ using importance re-sampling.

4 Performance

We now provide results of our theoretical and experimental analysis of the performance of the particle filter method. We also compare our implementation against the CUDA C implementation of Hurn et al. [5] to show consistency between OpenCL and CUDA for the same device.

4.1 Theoretical analysis

The most significant computational aspect of our approach is the evaluation of the log-likelihood function for a given set of model parameters. The number of double precision floating point operations (FLOPs) needed in this evaluation is

$$C_L = 1 + 2C_{\text{div}} + 7C_{\text{exp}} + C_F + C_S + C_I + N_t C_{PF}\,, \tag{17}$$

and

$$\begin{aligned}
C_F &= 1 + 544 D_m\,,\\
C_S &= 5 + C_{\text{div}} + C_{\text{sqrt}} + N_b\left(9 + C_{\text{sqrt}}\right),\\
C_{PF} &= N_p\left[2 + C_{\text{div}} + C_{\text{sqrt}} + C_{\text{log}} + N_s(15 + 2C_{\text{sqrt}})\right]\\
&\quad + N_p N_o\left[23 + 4C_{\text{div}} + 3C_{\text{log}} + C_{\text{exp}}\right.\\
&\quad \left. + N_f\left(30 + 2C_{\text{div}} + C_{\text{exp}} + 4C_{\text{trig}}\right)\right],
\end{aligned}$$

where C_F, C_S and C_{PF} are the FLOPs for Fourier coefficient calculation, particle filter initialisation (via burn-in simulations) and the particle filter, respectively; C_{div}, C_{sqrt}, C_{exp}, C_{log} and C_{trig} are the average number of FLOPs required for standard mathematical operations, D_m is the maximum days to maturity for an option, and N_o is the average number of options on an asset at any given time.

For our experiments we used $N_t = 4\,538$, $N_p = 32\,768$, $N_s = 4$, $N_o = 4$, $N_b = 3\,024$, $N_f = 200$, $D_m = 90$, $C_{\text{div}} \approx 2$, $C_{\text{sqrt}} \approx 3$, $C_{\text{exp}} \approx C_{\text{log}} \approx C_{\text{trig}} \approx 120$. An entire MLE for model parameters evaluates on average

Table 1: Theoretical performance of a MLE evaluation on CPU, GPU and MIC architectures.

Device	Model	$R_{\max}$	run time (mins)
Intel CPU	E5-2670	166 GFLOPs	83
Nvidia Tesla GPU	M2090	666 GFLOPs	21
Intel Xeon Phi	5110P	1 000 GFLOPs	14

11 log-likelihood calls. Thus, the total number of FLOPs is $\approx 11 \times C_L \approx 11 \times 7.6 \times 10^{13} = 8.36 \times 10^{14}$.

Using the theoretical peak performance $R_{\max}$ of an architecture we calculated the theoretical run time. This was done for an Intel E5-2670 (8 core), an Nvidia M2090 Tesla GPU (512 CUDA cores) and an Intel Xeon Phi 5110P Many-Integrated-Core (MIC) co-processor (60 core). These devices were selected as they are similar technology generations. Table 1 lists the results. The theoretical improvement of the GPU over a single CPU is far less than the common rhetoric of $10\times$–$100\times$.

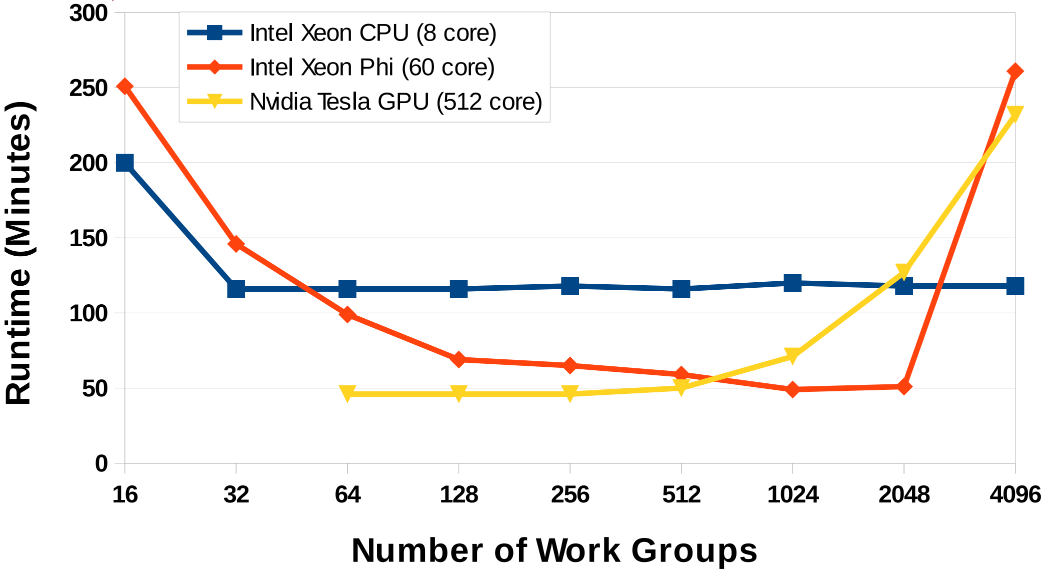
4.2 Experimentation

To simulate a fixed number of particles, several different work group sizes are possible. The optimum is dependent on the specific computation architecture used. We measured the average run times for our MLE implementation for work group sizes 2^m with $m \in [4, \ldots, 12]$.

Figure 1 gives the run times as a function of group size. The different devices have different behaviour; in particular, the Intel CPU is much less sensitive to the work group configuration. Run times should also be put into context with the original CUDA implementation which has an average run time of 48 minutes on the M2090 using 512 blocks (i.e., CUDA nomenclature for OpenCL work groups).

The run times in Figure 1 are also converted to an estimated measure of

Figure 1: Performance of the particle filtering algorithm using 32 768 particles for CPU, GPU and MIC devices.



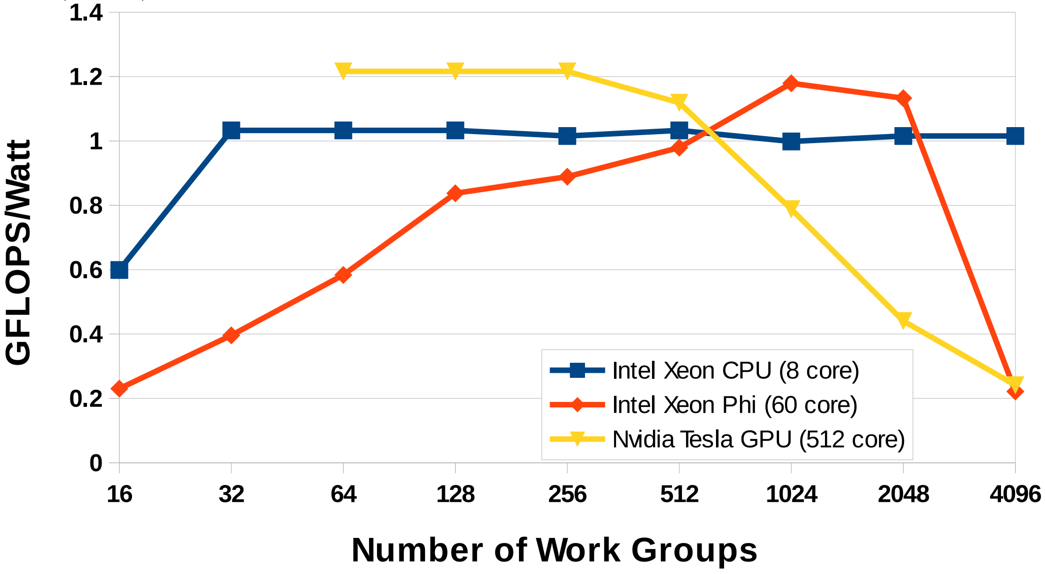
power consumption based on the maximum power output of the device and our estimated FLOP count in Section 4.1. The results, given in Figure 2, show that the difference in the maximum FLOPs per Watt for each of the devices is much less than the difference in raw compute times.

4.3 Evaluation

From our theoretical run time evaluations we evaluate how efficiently our OpenCL implementation utilises the available resources. Figure 3 shows this efficiency (i.e., the percentage of theoretical performance measured experimentally) as a function of work group size.

The GPU and MIC processors are severely under utilised with maximum efficiencies under 50% and 30%, respectively. There are many factors that are

Figure 2: GFLOPs/Watt for the particle method using 32 768 particles for CPU, GPU, and MIC devices.

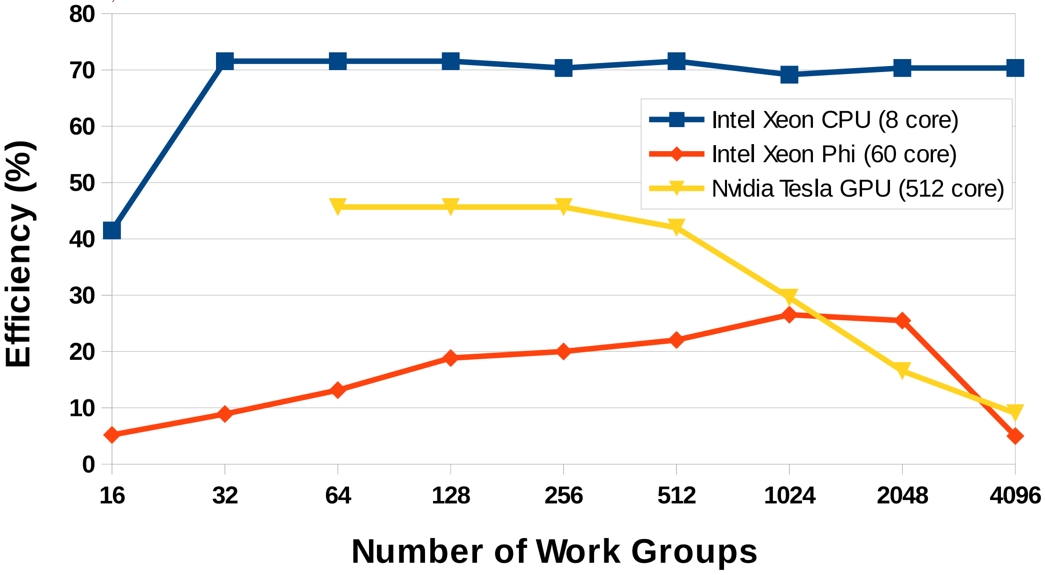


likely to be responsible for this, such as data transfer overheads over the PCI-e bus, or higher dependency on purely vectorised code and memory alignment. The more general purpose CPU is certainly less sensitive to such factors and this is reflected in the much higher efficiency measured.

From these experimental and theoretical results, we fairly compare the effectiveness of using accelerators for derivative pricing using stochastic volatility models, such as the Heston model. Theoretical analysis indicates that there is potential for up to $4\times$ to $6\times$ speed-up over a typical server CPU by using a GPU or MIC device. However, in practice, this speed-up is not readily attained by a direct implementation in CUDA C or OpenCL. Rather counter-intuitively, it would seem that such languages are more effective at utilising a standard CPU.

The observed efficiency of CPUs compared to GPUs is important in the context of many reported speed-ups from using GPUs. It indicates that such

Figure 3: Efficiency percentage of the particle method with 32 768 particles for CPU, GPU and MIC devices.



exaggerated claims as 100× [8] performance improvement are more indicative of forcing a streaming programming model (such as CUDA C or OpenCL) onto an algorithm as opposed to any underlying advantage of the GPU architecture. Our results indicate that GPUs and MIC processors can be used to achieve speed-up, but achieving more than 2.5× without significant attention to low-level optimisation is unlikely when the comparison is a fair one.

5 Conclusion

This article built upon the work of Hurn et al. [5, 6] to develop a multi-platform implementation of a maximum-likelihood estimator for the Heston stochastic volatility model using particle filtering and Fourier series approximations for derivative pricing. Our theoretical and experimental analysis evaluates the effectiveness of different computational architectures in terms of run time, FLOPs per Watt, and efficiency when compared with the theoretical FLOP counts. Our findings suggest that accelerators have the potential to provide modest speed-up, but the effort to reach this speed-up is high when fair comparisons are made. Furthermore, if speed-ups of the order of $100\times$ are required, then changing the programming model or improving the algorithm is more effective than targeting a specific device alone.

Acknowledgements This project utilised the high performance computing (HPC) facility at the Queensland University of Technology (QUT). The facility is administered by QUT's HPC and research support group. We also thank Dr Neil Kelson for presenting this work at EMAC2015 on our behalf.

References

[1] Y. Ait-Sahalia and R. Kimmel. Maximum likelihood estimation of stochastic volatility models. *J. Financ. Econ.*, 83:413–452, 2007. doi:10.1016/j.jfineco.2005.10.006 C365

[2] C. S. Forbes, G. M. Martin and J. Wright. Inference for a class of stochastic volatility models using option and spot prices: Application of a bivariate Kalman filter. *Economet. Rev.*, 26:387–418, 2007. doi:10.1080/07474930701220584 C368

[3] Khronos OpenCL Working Group. The OpenCL specification. Technical report, Khronos Group, October 2009. http://www.khronos.org/registry/cl/specs/opencl-1.0.pdf C372

[4] S. L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.*, 6:326–343, 1993. doi:10.1093/rfs/6.2.327 C366

[5] A. S. Hurn, K. A. Lindsay and A. J. McClelland. Estimating parameters of stochastic volatility models using option price data. *J. Bus. Econ. Stat.*, 33(4):579–594, 2015. doi:10.1080/07350015.2014.981634 C365, C366, C368, C370, C374, C378

[6] A. S. Hurn, K. A. Lindsay and A. J. McClelland. On the efficacy of Fourier series approximations for pricing European and digital options. *Appl. Math.*, 5(17):2786–2807, 2015. doi:10.4236/am.2014.517267 C366, C371, C378

[7] M. S. Johannes, N. G. Polson and J. R. Stroud. Optimal filtering of jump diffusions: Extracting latent states from asset prices. *Rev. Financ. Stud.*, 22:2759–2799, 2009. doi:10.1093/rfs/hhn110 C368

[8] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal and P. Dubey. Debunking the 100x GPU vs CPU myth: an evaluation of throughput computing on CPU and GPU. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pages 451–460, New York, NY, USA, 2010. ACM. doi:10.1145/1815961.1816021 C373, C378

[9] J. A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7(4):308–313, 1965. doi:10.1093/comjnl/7.4.308 C366

Author addresses

1. **A. S. Hurn**, School of Economics and Finance, Queensland University of Technology, Queensland 4001, Australia.
   mailto:s.hurn@qut.edu.au

2. **K. A. Lindsay**, School of Economics and Finance, Queensland University of Technology, Queensland 4001, Australia.
   mailto:kenneth.lindsay@glasgow.ac.uk

3. **D. J. Warne**, High Performance Computing and Research Support, Queensland University of Technology, Queensland 4001, Australia.
mailto:david.warne@qut.edu.au