広島経済大学研究論集
第37巻第2号　2014年9月

CUDA Fortran による確率的ボラティリティ変動モデルの GPU 計算

高　石　哲　弥*

1. はじめに

GPU は Graphics Processing Unit を表し，各種コンピュータの画像処理を行う主要部品の1つである。パソコンではグラフィックカードやビデオ―カードと知られる部品に付属していることが多い。近年の GPU は計算コアを多数持つようになっており，非常に計算能力が高くなっている。この計算能力の高さを利用して画像処理以外の目的に利用した計算が GPU 計算である。

GPU 計算が計算能力を発揮するのは多数の計算コアが同時に実行される並列計算においてである。現在は並列計算が可能な計算流体力学，天文シミュレーション，生命科学などの様々な分野で GPU 計算が行われている。本研究では，金融時系列モデルの推定に GPU 計算を応用する。金融時系列モデルの推定では，時系列を表すモデルを仮定し，そのモデルのパラメータを推定する。仮定するすべてのモデル推定で並列計算が有効となる訳ではない。本研究では実現確率的ボラティリティ変動（Realized Stochastic Volatility: RSV）モデル[1] の推定で GPU 計算を実行する。RSV モデルの推定方法はいくつかあるが[2]，ハイブリッドモンテカルロ法[3] を利用した場合，並列化が可能であり GPU 計算を行うことができる。

本研究では，PGI 社の CUDA Fortran[4] を用いて GPU 計算を行い，パソコンの CPU と計算速度を比較する[5]。Fortran は科学計算でよく利用されており，CUDA Fortran を用いることによって過去の Fortran プログラムからの移植が容易となる。本研究では Takaishi（2013）[6] で開発した Fortran プログラムをもとに CUDA Fortran プログラムを記述した。また，CUDA Fortran では，ディレクティブベースでプログラミングができる OpenACC もサポートしているので，OpenACC による GPU 計算も実行しその結果を比較する。

2. 実現確率的変動ボラティリティモデル（RSV モデル）

近年，株価の高頻度データが利用できるようになり，日中の株価収益率のデータから求める実現ボラティリティ[7] が計算されるようになっている。t 日における日中の収益率のデータがあるサンプリング間隔サンプルされ n 個あり，$(r_{t,1}, r_{t,2}, \cdots, r_{t,n})$ で表されるとすると，実現ボラティリティ RV_t は収益率の2乗の和，$\sum_{i=1}^{n} r_{t,i}^2$ で求められる。理論的にはサンプリング間隔がゼロの極限で実現ボラティリティは積分ボラティリティに収束するが，実際のデータはいくつかのバイアスを含んでおり，それらが真の実現ボラティリティからのズレをもたらす。RSV モデルは，日次収益率時系列データを用いる従来の SV モデルに実現ボラティリティの情報を取り入れることによって，モデルの精度を高めたモデルである。また，観測される実現ボラティリティにはバイアスあるが，RSV モデルはバイア

* 広島経済大学経済学部教授

スの影響も考慮することができるようになっている。本研究では，Takahashi et al.（2009）[1] によって定式化された以下の RSV モデルを利用する。

$$\begin{aligned} y_{1,t} &= \sigma_t \varepsilon_t, & \varepsilon_t &\sim N(0,1) \\ y_{2,t} &= \xi + h_t + u_t, & u_t &\sim N(0,\sigma_u^2) \\ h_{t+1} &= \mu + \phi(h_t - \mu) + \eta_t, & \eta_t &\sim N(0,\sigma_\eta^2) \\ h_1 &= \mu + \eta_0, & \eta_0 &\sim N\left(0, \frac{\sigma_\eta^2}{1-\phi^2}\right) \end{aligned} \quad (1)$$

ここで，$y_{1,t}$ は t 日の日次収益率，また $y_{2,t}$ は t 日の RV の対数値，$\ln RV_t$ を表している。ボラティリティ変数 h_t は $h_t = \ln \sigma_t^2$ で定義される。このモデルにおけるパラメータは $(\phi, \mu, \xi, \sigma_u^2, \sigma_\eta^2)$ の５つあり，これらのパラメータを与えられた時系列データのもとで推定することになる。

一般に，モデルのパラメータ推定では，モデルの尤度関数を求め，尤度関数を最大化するパラメータ値が推定値として採用されることが多い。このモデルの尤度関数 $L(Y_1, Y_2 \mid \theta)$ は以下である。

$$\begin{aligned} L(Y_1, Y_2 \mid \theta) &= \int f(Y_1, Y_2 \mid \theta, h)\, dh_1 \cdots dh_T, \\ f(Y_1, Y_2 \mid \theta, h) &= \prod_{t=1}^{T} (2\pi)^{-1/2} e^{-h_t/2} \exp\left(-\frac{y_{1,t}^2}{2e^{h_t}}\right) \\ &\quad \times \left(2\pi\sigma_u^2\right)^{-1/2} \exp\left(-\frac{(y_{2,t} - \xi - h_t)^2}{2\sigma_u^2}\right) \\ &\quad \times \exp\left(-\frac{(h_2 - \mu)^2}{2(\sigma_\eta^2/(1-\phi^2))}\right) \prod_{t=2}^{T} \left(2\pi\sigma_u^2\right)^{-1/2} \\ &\quad \times \exp\left(-\frac{(h_{t+1} - \mu - \phi(h_t - \mu))^2}{2\sigma_\eta^2}\right) \end{aligned} \quad (2)$$

ここで，θ, h, Y_1, Y_2 はそれぞれ $\theta \equiv (\phi, \mu, \xi, \sigma_u^2, \sigma_\eta^2)$，$h = h_1, h_2, \cdots, h_T$　$Y_1 = y_{1,1}, y_{1,2} \cdots, y_{1,T}$，$Y_2 = y_{2,1}, y_{2,2} \cdots, y_{2,T}$ を表す。また，T は時系列長である。尤度関数 $L(Y_1, Y_2 \mid \theta)$ は積分形のため，その値を評価するのが難しく，最尤法には適していない。そのため，SV 型モデルでは最尤法ではなく，ベイズ推定をもとにしたマルコフ連鎖モンテカルロ法が利用されることが多い。

ベイズ推定ではパラメータの従う確率分布が以下のように与えられる。

$$P(\theta \mid Y_1, Y_2) = \frac{L(Y_1, Y_2 \mid \theta)\pi(\theta)}{f(Y_1, Y_2)} \quad (3)$$

ここで，$\pi(\theta)$ はパラメータの事前分布，$f(Y_1, Y_2)$ は規格化定数にあたる。本研究では，事前分布は定数と仮定する。定数部分を c と置くと，$P(\theta \mid Y_1, Y_2) = cL(Y_1, Y_2 \mid \theta)$ と書ける。パラメータの推定値はこの確率分布のもとでの期待値として得られる。

$$\langle \theta \rangle = c \int \theta f(Y_1, Y_2 \mid \theta, h)\, dh_1 \cdots dh_T d\theta \quad (4)$$

この積分は一般には，解析的に実行できないので，マルコフ連鎖モンテカルロ法によって期待値を見積もる。積分にはパラメータの積分と T 個のボラティリティ変数の積分があり，それぞれマルコフ連鎖モンテカルロ法を実行しなければならない。本研究では，T 個のボラティリティ変数に対しては並列に実行できるハイブリッドモンテカルロ法を用いる。

3.　ハイブリッドモンテカルロ法

ハイブリッドモンテカルロ法[3] は分子動力学シミュレーションとメトロポリス法の組み合わせから成る。ハイブリッドモンテカルロ法の特徴の１つは，多数の変数を同時にサンプルすることが可能なことである。分子動力学シミュレーションではハミルトン方程式を数値的に解いて時間発展させる。ハミルトン方程式に現れるハミルトニアンは，もともとは存在しないが，運動量変数を導入して定義する。

ハミルトニアンは以下のように定義される。ボラティリティ変数 $h = (h_1, h_2, \cdots, h_T)$ に共役な運動量 $p = (p_1, p_2, \cdots, p_T)$ を導入して（4）式は

$$\begin{aligned}\langle\theta\rangle &= c\int \theta f(Y_1,Y_2\,|\,\theta,h)dh_1\cdots dh_T d\theta \\ &= c\int \theta \exp(\ln f(Y_1,Y_2\,|\,\theta,h))dh_1\cdots dh_T d\theta \\ &= \frac{1}{Z}\int \theta \exp\left(-\frac{1}{2}\sum_{i=1}^{T} p_i{}^2 + \ln f(Y_1,Y_2\,|\,\theta,h)\right) \\ &\quad \times dp_1\cdots dp_T dh_1\cdots dh_T d\theta \\ &= \frac{1}{Z}\int \theta \exp(-H(p,h))dp_1\cdots dp_T dh_1\cdots dh_T d\theta\end{aligned} \tag{5}$$

と書くことができる。ここで，Z は規格化定数であり，定数 c は Z の中に再定義されている。また，ハミルトニアン $H(p,h)$ は

$$H(p,h) = \frac{1}{2}\sum_{i=1}^{T} p_i{}^2 - \ln f(Y_1,Y_2\,|\,\theta) \tag{6}$$

で定義される。Z は運動量積分を 1 にするための規格化定数であるが，マルコフ連鎖モンテカルロ法を実行するうえでは重要ではない。また，運動量の導入によってパラメータの期待値が変化することはない。

（5）式をマルコフ連鎖モンテカルロ法によって実行する際に，ボラティリティ変数は $\exp(-H(p,h))$ に比例する確率で生成することになる。ハイブリッドモンテカルロ法では，新たなボラティリティ変数の候補をハミルトン方程式にしたがって時間発展させることによって得る。ハミルトン方程式は以下で表わされる。

$$\begin{cases}\dfrac{dh_i}{d\tau} = \dfrac{\partial H(p,h)}{\partial p_i} \\ \dfrac{dp_i}{d\tau} = -\dfrac{\partial H(p,h)}{\partial h_i}\end{cases} \tag{7}$$

ここで，τ はハミルトン方程式における仮想の時間であって実際の時間ではない。一般にこれらの方程式は解析的には解けないので，分子動力学シミュレーションによって数値的に小さな時間ステップサイズ $\Delta\tau$ で積分して時間発展させる。近似的に時間発展させるので，ステップサイズ $\Delta\tau$ による誤差が入ってくることになる。従って，ハミルトン方程式はハミルトニアンを保存するが，分子動力学シミュレーションによって時間発展させた後のハミルトニアンは保存せず，ステップサイズ $\Delta\tau$ に応じたズレが生じる。近似的に積分するすべての方法がハイブリッドモンテカルロ法で利用できるわけではない。本研究ではハイブリッドモンテカルロ法で利用できる条件を備え，且つよく利用されている以下の Leapfrog 法を用いる[8]。

$$\begin{cases}h_i(\tau+\Delta\tau/2) = h_i(t+) + p_i(\tau)\Delta\tau/2 \\ p_i(\tau+\Delta\tau) = p_i(\tau) - \dfrac{\partial H}{\partial h_i}\Delta\tau \\ h_i(\tau+\Delta\tau) = h_i(t+) + p_i(\tau+\Delta\tau)\Delta\tau/2\end{cases} \tag{8}$$

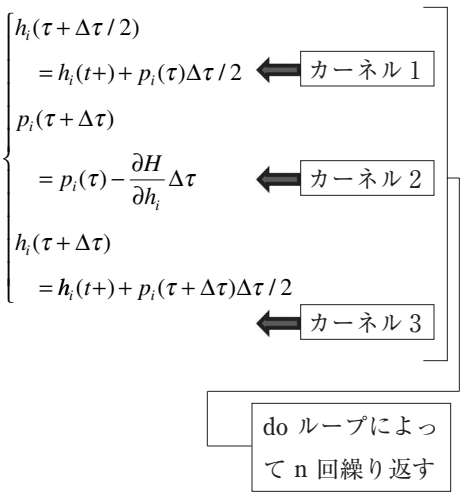
分子動力学シミュレーションでは（8）式に従ってすべてのボラティリティ変数及び運動量を積分してある時間まで時間発展させる。時間発展させたあとのハミルトニアンは $O(\Delta\tau^2)$ の誤差となる。

積分によって得られた新たな運動量とボラティリティ変数を (p',h') とする。この時のハミルトニアンは初めのハミルトニアンと $\Delta H = H(p',h') - H(p,h)$ だけ違っている。この ΔH を用い，新たなボラティリティ変数 h' をメトロポリス法によって以下の確率で選択する。

$$P = \min(1, \exp(-\Delta H)) \tag{9}$$

4. CUDA Fortran によるプログラム

本研究では（8）式を GPU 計算によって並列計算する。CUDA Fortran では以下のように 3 つのカーネルを実行し，ステップサイズ $\Delta\tau$ の積分を実行する。そして，n 回繰り返すことによって，$l = n\times\Delta\tau$ までの積分が終了する。



３つのカーネルの内，カーネル１とカーネル３は同じプルグラムから成り立つ。

上記の部分に関する CUDA　Fortran での大まかなプログラムは以下となる。

```
real(4), allocatable,device:: d_h(:),d_p(:),
d_dy(:),d_grv(:)
ngrid = (nd-1)/thread + 1; nd は時系列データの長さ, thread はスレッド数

allocate(d_h(nd),d_p(nd),d_dy(nd),d_grv
(nd)); 変数 h,p 及び Y1, Y2 のデバイス配列の確保

do i=1, n
  call steph<<<ngrid,thread>>>(nd,d_h,d_p,
  dt); カーネル 1 の呼び出し
call stepp<<<ngrid,thread>>>(nd,d_h,d_p,d_
dy,d_grv,dt,theta); カーネル 2 の呼び出し
  ここで，dt は Δτ，theta はパラメータ値の入った配列である。
call steph<<<ngrid,thread>>>(nd,d_h,d_p,
dt); カーネル 3 の呼び出し

enddo
```

デバイスコードはカーネル１に対応するコードのみを以下に示しておく。

```
attributes(global) subroutine steph(nd,d_h,
d_p,dt)
implicit none
integer,value :: nd
real(4), value :: dt
real(4),device :: d_h(nd),d_p(nd)
integer :: j

j=(blockIdx%x–1)* blockDim%x+threadIdx%x
if ( j< nd +1) d_h(j)=d_h(j)+d_p(j)*dt*0.5
return
end
```

5.　開　発　環　境

本研究では，Intel i7-4770 3.4GHz CPU を搭載したパソコンを利用し，PGI 社の Windows 版 Fortran コンパイラ（PGI 14.6）をインストールした。PGI 社の Fortran コンパイラは CUDA Fotran 及び OpenACC が利用可能である。CUDA ドライバーのバージョンは6.0を利用した。GPU には NVIDIA 社の GeForce GTX 760 を利用した。表１は GeForce GTX 760 の基本スペックをまとめたものである。

表１　GeForce GTX 760のスペック

GPU エンジンスペック	
CUDA コア	1,152個
ベースクロック	980 MHz
ブーストクロック	1,033 MHz
メモリースペック	
メモリクロック	3,004 MHz
メモリ量	2,048 MB
最大バンド幅	192.2 GB/sec
メモリインターフェース	256-bit GDDR5

6. 計算時間の比較

本研究では，時系列データの長さ T を T=512×B として，時系列データ長を変化させて GPU 及び CPU での計算時間を比較する。GPU 計算のとき，スレッド数は512とし，ブロック数 B を変化させた。計算時間は（8）式，即ち Leapfrog 法に対応する部分を10,000回繰り返し，Leapfrog 法 1 回分の計算時間の平均値として求めた。図 1 は GPU と CPU での計算時間を B の関数として表したものである。計算時間は GPU，CPU 共に B に比例して増加しているが，GPU の方が計算時間はゆっくりと増加している。表 2 は計算時間を f(B)＝A＋C*B とおき，B の関数としてフィティングを実行した係数 A と C の結果である。

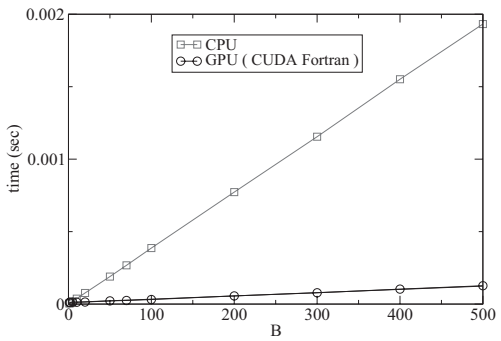


図 1　GPU（○）と CPU（□）による計算時間
計算時間は Leapfrog 法 1 回当たりの平均時間である。

表 2　フィッティングによる関数 f(B)＝A＋C*B の係数 A と C

	A	C
CPU	–1.42e–6	3.87e–6
GPU（CUDA Fortran）	1.13e–5	2.25e–7

図 2 はフィティングした関数を利用して，CPU に対する GPU の高速化率（Gain）を Gain＝(CPU の計算時間)／(GPU の計算時間) として求めたものである。図 3 は図 2 と同じであるが，B<50の部分のみを表示したものである。図 3 から，B が 3 より小さい場合は Gain が 1 以下で，これは CPU 計算よりも計算速度が遅いことを意味しているが，B が大きくなると GPU 計算の計算速度が速くなる。B が無限大の極限，つまり時系列長が無限大の極限で Gain は表 2 の C の係数から予想される Gain＝3.87e–6/2.25e–7＝17.2になる考えられる。

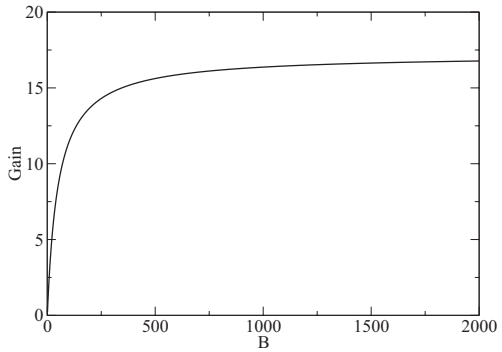


図 2　CPU に対する GPU での計算の高速化率

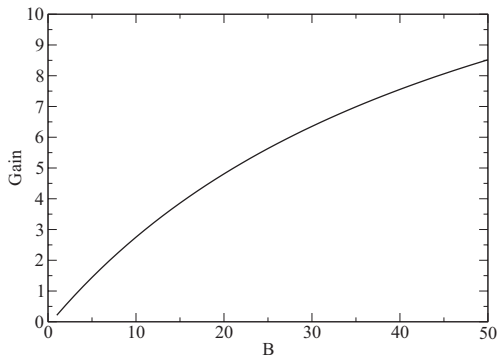


図 3　CPU に対する GPU での計算の高速化率
図 2 において B <50を拡大した表示したものである。

7. OpenACC による GPU 計算

OpenACC は既存のプログラムにディレクティブを挿入することによって GPU 計算用のコードを自動生成する。従って，CUDA Fortran よりも簡便に GPU 計算用プログラムを記述することができる。以下は，kernels ディレ

クティブを Leapfrog 法の計算のために挿入した所を表している。kernels ディレクティブに囲まれた部分の GPU カーネルが生成される。

!$acc kernels ← ディレクティブの挿入

$$\begin{cases} h_i(\tau+\Delta\tau/2) = h_i(t+) + p_i(\tau)\Delta\tau/2 \\ p_i(\tau+\Delta\tau) = p_i(\tau) - \dfrac{\partial H}{\partial h_i}\Delta\tau \\ h_i(\tau+\Delta\tau) \\ \quad = \boldsymbol{h}_i(t+\Delta\tau/2) + p_i(\tau+\Delta\tau)\Delta\tau/2 \end{cases}$$

!$acc end kernels ← ディレクティブの挿入

do ループによって n 回繰り返す

図 4 の○は kernels ディレクティブを上記のように挿入した場合の計算時間を表しているが，CPU 計算と比較して高速化が図られていないばかりか，B の小さいところでは CPU よりも計算時間が掛かる結果となっている。これは，kernels ディレクティブでは，ホスト―デバイス間のデータ転送は自動で行われるので，必要のないデータ転送によって計算時間がかかっていることを示している。本研究の場合，ボラティリティ変数 h と運動量変数 p は Leapfrog 法 1 回毎にデータ転送する必要はなく，do ループの前に 1 度転送するだけで良い。この場合，以下のように data ディレクティブを挿入することによってデータ転送を制御することができる。

!$acc data copy(h,p) ← data ディレクティブの挿入

!$acc kernels

$$\begin{cases} h_i(\tau+\Delta\tau/2) = h_i(t+) + p_i(\tau)\Delta\tau/2 \\ p_i(\tau+\Delta\tau) = p_i(\tau) - \dfrac{\partial H}{\partial h_i}\Delta\tau \\ h_i(\tau+\Delta\tau) = \boldsymbol{h}_i(t+) + p_i(\tau+\Delta\tau)\Delta\tau/2 \end{cases}$$

!$acc end kernels

!$acc end data ← data ディレクティブの挿入

do ループによって n 回繰り返す

図 4 の□は data ディレクティブを挿入した後の計算時間を表している。data ディレクティブを挿入することによって CUDA Fortran と同等の計算速度となっている。

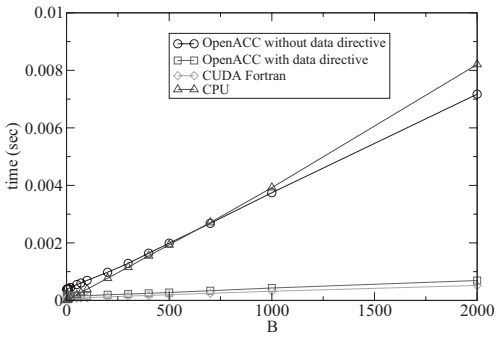


図 4　OpenACC を利用した時の計算時間と CPU と GPU（CUDA Fortran）の計算時間の比較

8. ま　と　め

確率的ボラティリティ変動モデルのベイズ推定において利用されるハイブリッドモンテカルロ法は並列化可能である。本研究では，GPU 計算によってハイブリッドモンテカルロ法の並列計算を行った。GPU コードは PGI 社の CUDA Fortran によって開発した。GPU と CPU のそれぞれで計算時間を測定し比較したところ，時

系列データ長が長いほど CPU に対する GPU の計算速度が速くなり，時系列長が無限大の極限で約17倍速くなることがわかった。

また，ディレクティブを挿入することによって容易に GPU 計算を実行できる OpenACC よる GPU 計算を行った。そして，適切なディレクティブ挿入を行うことによって CUDA Fortran と同様の計算速度を達成できることが分かった。

付記：本研究は平成25年度広島経済大学特定個人研究費による助成を受けたものである。

注

1） M. Takahashi, Y. Omori and T. Watanabe, Estimating stochastic volatility models using daily returns and realized volatility simultaneously, Compt. Stat. & Data anal. 53（2009）2404–2425.
2） SV モデルの一般的な推定方法については次を参照。渡部敏明，ボラティティ変動モデル，朝倉書店（2000）.
3） S. Duane et. al, Hybrid Monte Carlo, Physics Letters B195（1987）216–222.
4） PGI CUDA Fortran の日本での販売会社は（株）ソフテックである。http://www.softek.co.jp/SPG/Pgi/index.html
5） CUDA C での結果については次を参照。高石哲弥，GPGPU によるストカスティックボラティリティモデルのベイズ推定，広島経済大学研究論集第33巻第 1 号21–27.
6） T. Takaishi, Bayesian estimation of realized stochastic volatility model by Hybrid Monte Carlo algorithm, Journal of Physics: Conference Series 490（2014）012092.
7） T. G. Andersen and T. Bollerslev, Answering the Skeptics: Yes, Standard Volatility Models Do Provide Accurate Forecasts, International Economic Review 39（1998）885–905.; T. G. Andersen, T. Bollerslev, F. X. Diebold and H. Ebens, The Distribution of Realized Stock Return Volatility, Journal of Financial Economics 61（2001）43–76.
8） Leapfrog 法以外の積分法については，例えば次を参照。T. Takaishi, Choice of Integrator in the Hybrid Monte Carlo Algorithm, Computer Physics Communications 133（2000）6–17.; T. Takaishi, Higher Order Hybrid Monte Carlo at Finite Temperature, Physics Letters B540（2002）159–165.; T. Takaishi and Ph. de Forcrand, Testing and tuning symplectic integrators for Hybrid Monte Carlo algorithm in lattice QCD, Physical Review E73（2006）036706.