Parallelization strategies for continuum-generalized method of moments on the multi-thread systems

Cite as: AIP Conference Proceedings 1862, 030146 (2017); https://doi.org/10.1063/1.4991250
Published Online: 10 July 2017

A. Bustamam, T. Handhika, Ernastuti, and D. Kerami

ARTICLES YOU MAY BE INTERESTED IN

Data preprocessing for determining outer/inner parallelization in the nested loop problem using OpenMP
AIP Conference Proceedings 1862, 030138 (2017); https://doi.org/10.1063/1.4991242

Hypergraph partitioning implementation for parallelizing matrix-vector multiplication using CUDA GPU-based parallel computing
AIP Conference Proceedings 1862, 030153 (2017); https://doi.org/10.1063/1.4991257

Analysis of Indonesian educational system standard with KSIM cross-impact method
AIP Conference Proceedings 1862, 030135 (2017); https://doi.org/10.1063/1.4991239
Parallelization Strategies for Continuum-Generalized Method of Moments on the Multi-Thread Systems

A. Bustamam¹, T. Handhika² a), Ernstutti², and D. Kerami¹

¹Department of Mathematics, Faculty Mathematics and Natural Sciences (FMIPA), Universitas Indonesia, Depok 16424, Indonesia
²Computational Mathematics Study Center, Gunadarma University, Depok, Indonesia

a)Corresponding author: trihandika@staff.gunadarma.ac.id

Abstract. Continuum-Generalized Method of Moments (C-GMM) covers the Generalized Method of Moments (GMM) shortfall which is not as efficient as Maximum Likelihood estimator by using the continuum set of moment conditions in a GMM framework. However, this computation would take a very long time since optimizing regularization parameter. Unfortunately, these calculations are processed sequentially whereas in fact all modern computers are now supported by hierarchical memory systems and hyperthreading technology, which allowing for parallel computing. This paper aims to speed up the calculation process of C-GMM by designing a parallel algorithm for C-GMM on the multi-thread systems. First, parallel regions are detected for the original C-GMM algorithm. There are two parallel regions in the original C-GMM algorithm, that are contributed significantly to the reduction of computational time: the outer-loop and the inner-loop. Furthermore, this parallel algorithm will be implemented with standard shared-memory application programming interface, i.e. Open Multi-Processing (OpenMP). The experiment shows that the outer-loop parallelization is the best strategy for any number of observations.

INTRODUCTION

The uncertainty of the real world often causes a problem be modeled as a function of the number of unknown parameters. These parameters need to be estimated in advance so it is useful in understanding the behavior of related issues. This happens also in the field of finance, including asset pricing. There are several statistical methods that can be used to estimate the parameter of a model, for example, Maximum Likelihood Estimation (MLE) and Generalized Method of Moments (GMM). MLE requires information regarding probability density function (pdf) that makes the estimator become efficient. In fact, pdf is frequently not available in analytical form. It is different with a characteristic function which is used in GMM. But, the efficiency of GMM estimator depends on the appropriate set of moment conditions. Accordingly, Marine Carrasco and Jean-Pierre Florens [1] developed a method which combines the attractive features of GMM with the efficiency of MLE in one framework that was called Continuum-Generalized Method of Moments (C-GMM) to rely on a continuum of moment conditions in a GMM procedure. To improve the objectivity of the C-GMM method, it is required optimization stages to the new parameter, known as regularization parameter [2]. It is optimized through two-stage estimation in which the optimal regularization parameter is obtained by determining the regularization parameter that makes the Mean-Squared of Error (MSE) of the C-GMM estimator based on sampled simulation become minimum [3]. This, of course, takes a very long time, especially in the calculation which is processed sequentially. This increases the latency of calculation process, particularly impact on the arbitrage problem in asset pricing [4, 5, 6]. Meanwhile, all modern computers are now supported by hierarchical memory systems and hyperthreading technology, which allowing for parallel computing.

Parallel computing uses multiple compute resources simultaneously to solve a computational problem that is divided into several independent discrete tasks which can be solved concurrently on different processors [7]. Figure 1 illustrates a comparison of the time required between sequential and parallel calculation process for a problem that
can be divided into \( n \) independent tasks. It is seen that parallel computing with multiple processors is faster than sequential computing with only using one processor. Prior to 1990, Intel had put a million transistors onto a single chip. Since then the computer technology continues to evolve, especially in the high-performance computing (HPC). If some of Central Processing Units (CPU) previously behaved as shared-memory parallel machines, it is now only needed a single CPU with multi-core to perform parallelization by replicating substantial parts of a processor's logic on a single chip. Memory distribution constraints in the multi-core configuration then caused a computer was built with hierarchical memory systems that supply the processor with data and instructions at high rates by using a small, expensive and very fast memory called cache memory [8]. In addition to hierarchical memory systems, Intel's hyperthreading technology also improves the calculation performance by using thread concept. A thread is a runtime entity which is able to independently execute a stream of instructions that only needs its program counter and an area in memory to save its variables including registers and a stack [8]. Multiple threads may be executed on a single core even on multiple cores via context switches.

The multi-thread systems start sequentially as a single-thread of execution, known as the initial thread. Afterward, it creates a team of threads for starting the parallelization. The initial thread becomes the master of the team and collaborates with the others to execute the code dynamically. Finally, they are joined to the only one thread (the initial thread) while all others terminate. Unfortunately, a phenomenon that occurs at this time is there are many programmers who still build a program sequentially on the parallel computer architecture. They do not take advantage of this HPC technology to improve the performance of their calculation. This paper aims to design a parallel algorithm for C-GMM on the multi-thread systems for reducing the latency to approximate real-time problem.

**CONTINUUM-GENERALIZED METHOD OF MOMENTS**

This paper restricts the attention to a random vector Markov process, \( x(t) \in \mathbb{R}^p \), whose distribution is indexed by a finite dimensional parameter \( \vartheta \in \mathbb{R}^n \) with true value \( \vartheta_0 \). The moment function of this random vector Markov process is as follow:

\[
h_t(\tau, \vartheta) = \left( e^{i\tau^T x(t)} - \psi_{x(t)}(\tau_2, \vartheta) \right) e^{i\tau_1^T x(t)}, \tag{1}\]

where \( \psi_{x(t)}(\tau_2, \vartheta) \) is a conditional characteristic function of \( x(t) \) and \( \tau = (\tau_1, \tau_2) \in \mathbb{R}^{2p} \). Next, let \( \pi \) be a probability density function (pdf) on \( \mathbb{R}^{2p} \) and \( L^2(\pi) \) be the Hilbert space of complex valued functions that are square integrable with respect to \( \pi \), i.e. [3]:

\[
L^2(\pi) = \{ f : \mathbb{R}^{2p} \rightarrow \mathbb{C} | \int f(\tau)\bar{f}(\tau)\pi(\tau) d\tau < \infty \},
\]

where \( \bar{f}(\tau) \) denotes the complex conjugate of \( f(\tau) \). By considering:

\[
(f, g) = \int f(\tau)\bar{g}(\tau)\pi(\tau) d\tau,
\]

with \( \| f \|^2 = (f, f) \) and \( \tilde{h}_{\tau}(\tau, \vartheta) = \frac{1}{\tau} \sum_{t=1}^{\tau} h_t(\tau, \vartheta) \), then the efficient C-GMM estimator is given by [1]:

![FIGURE 1. The Comparison of Computational Time: Sequential VS Parallel Computing](image-url)
\[ \hat{\theta}_T = \arg \min_{\theta} (K^{-1} \hat{h}_T(\tau, \theta), \tilde{h}_T(\tau, \theta)), \]

where \( K \) is the Hilbert-Schmidt integral operator which is an asymptotic covariance operator associated with the moment conditions and satisfies:

\[ Kf(\tau) = \int_{-\infty}^{\infty} k(\tau, \theta) f(\tau_2) \pi(\tau_2) d\tau_2, \]

where \( k(\tau, \theta) \) is the kernel defined by \( k(\tau, \theta) = E[h_t(\tau_1, \theta) h_t(\tau_2, \theta)] \). This kernel can be estimated by \( \tilde{h}_T(\tau, \theta^1) = \frac{1}{T} \sum_{t=1}^{T} h_t(\tau_t, \theta^1) \bar{h}_t(\tau_t, \theta^1) \) with a sample of size \( T \) where \( \theta^1 \) is a consistent first step estimator \([3]\). Hence, the estimated Hilbert-Schmidt integral operator is denoted by \( K_T \). Unfortunately, \( K_T \) is singular such that the \( K_T^{-1} \) needs to be estimated by \( K_T^{-1} = (K_T^2 + \alpha I)^{-1} K_T \) which involves a new regularization parameter, denoted by \( \alpha \).

The regularization parameter should be conducted independently in each trial in order to prevent over-fitting in model selection bias \([9]\). Therefore, the feasible efficient C-GMM estimator is given by:

\[ \hat{\theta}_T(\alpha) = \arg \min_{\theta} (K^{-1}_T \hat{h}_T(\tau, \theta), \tilde{h}_T(\tau, \theta)), \]

where \( \pi \) is a pdf of random variables which have standard bivariate normal distribution. In most real-life problems, \( \Sigma \) is unknown, then it needs to be estimated via parametric bootstrap:

\[ \hat{\theta}_T(\alpha) = \arg \min_{\theta} \Sigma_T(\alpha, \theta_0), \]

where \( \Sigma_T(\alpha, \theta_0) = TE(\|\hat{\theta}_T(\alpha) - \theta_0\|^2) \). This raises problems related to possible infiniteness of \( \Sigma_T(\alpha, \theta_0) \) and unknown distribution of \( \hat{\theta}_T(\alpha) - \theta_0 \) for the finite sample. To hedge against such situations, considering optimal regularization parameter as \([3]\):

\[ \alpha_T(\theta_0) = \arg \min_{\alpha \in [0,1]} \Sigma_T(\alpha, \theta_0), \]

which defined as a C-GMM estimator of \( \theta_0 \).

By using \( \hat{\theta}_T^J(\hat{\theta}_T^J) \) to simulate \( M \) independent samples of size \( T \) from \( M \) arbitrary initial values \( \hat{\theta}_0^{(j)} \), then be obtained \( X_t^{(j)}(\hat{\theta}_T^J) \) for \( j = 1, 2, \cdots, M \) which satisfies \( X_t^{(j)}(\hat{\theta}_T^J) = f(X_t^{(j)}, \hat{\theta}_T^J, \epsilon_t^{(j)}) \) for \( t = 1, 2, \cdots, T \) with \( f(\cdot) \) is a three times continuously differentiable function with respect to \( \hat{\theta}_T^J \) and \( \epsilon_t^{(j)} \) are independent and identically distributed white noise whose distribution are known and do not depend on \( \hat{\theta}_T^J \). Based on each simulation, then estimate the parameter \( \theta \) by using \( \hat{\theta}_T^{(j)}(\alpha, \theta_0^j) \) which is denoted as a C-GMM estimator at \( J \) with \( \alpha \)-th regularization parameter from \( j \)-th sample for any value of regularization parameter that is evaluated. Hence, the optimal regularization parameter at equation \( 3 \) is obtained by selecting a point in the grid for regularization parameter which satisfies:

\[ \alpha_{TM}(\hat{\theta}_T^J) = \arg \min_{\alpha \in [0,1]} \Sigma_{TM}(\alpha, \hat{\theta}_T^J), \]

where \( \Sigma_{TM}(\alpha, \hat{\theta}_T^J, v) \) is the truncated MSE estimator of \( \Sigma_T(\alpha, \theta_0, v) \) in equation \( 4 \) which is given by:

\[ \Sigma_{TM}(\alpha, \hat{\theta}_T^J, v) = \frac{1}{M} \sum_{j=1}^{M} \xi_{j,T}(\alpha, \hat{\theta}_T^J) \mathbf{1}(\xi_{j,T}(\alpha, \hat{\theta}_T^J) \leq \tilde{n}_v) + v \tilde{n}_v T, \]

where

\[ \frac{1}{M} \sum_{j=1}^{M} \mathbf{1}(\xi_{j,T}(\alpha, \hat{\theta}_T^J) \leq \tilde{n}_v) = 1 - v \text{ and } \xi_{j,T}(\alpha, \hat{\theta}_T^J) = \|\hat{\theta}_T^{(j)}(\alpha, \theta_0^j) - \hat{\theta}_T^J\|^2. \]

For some assumptions, using \( \alpha_{TM}(\hat{\theta}_T^J) \) at \( 6 \) to estimate \( \hat{\theta}_T(\alpha) \) in equation \( 2 \) does not affect the consistency, asymptotic normality, and efficiency of the C-GMM estimator \([3]\). Therefore, the feasible efficient C-GMM estimator which depends on the optimal regularization parameter is defined as follow:

\[ \hat{\theta}_T(\alpha_{TM}) = \arg \min_{\theta} (K^{-1}_{\alpha_{TM}} \hat{h}_T(\tau, \theta), \tilde{h}_T(\tau, \theta)). \]

**PARALLELIZATION STRATEGIES**

In the previous section, it has been discussed briefly how to estimate the parameter model by using the C-GMM method. It is seen that the manual calculation process is extremely complex so it needed a tool such as a computer
for programming this method. But earlier, the C-GMM method had to be written back into a structured and systematic sequential algorithm such that minimizing programming error. To understand parallelization strategies for the sequential C-GMM algorithm, the algorithm will be written in terms of vectors and matrices.

Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ be a vector of the grid for regularization parameter with size $N$. Besides, $M$ independent simulated samples of size $T$ is given by:

$$X_T^M = \begin{pmatrix}
X_1^{(1)} & X_1^{(2)} & \cdots & X_1^{(M)} \\
X_2^{(1)} & X_2^{(2)} & \cdots & X_2^{(M)} \\
\vdots & \vdots & \ddots & \vdots \\
X_T^{(1)} & X_T^{(2)} & \cdots & X_T^{(M)}
\end{pmatrix}$$

where $x_k^{(j)}$ be denoted $j$-th simulated sample for $k$-th time. As mentioned earlier, $X_T^{(j)}$ denotes a $j$-th column vector of $X_T^M$ which defined as $j$-th simulated sample of size $T$. Hence, the original C-GMM algorithm is given in Fig. 2.

In this paper, the optimal parameters are obtained through the Nelder-Mead algorithm [10] which combined with numerical integration. The variables that are included consist of the number of observations and simulations, initial guess for parameter model, the maximum iteration, and tolerance in the optimization stages, the interval values of the grid for $\tau$ and the regularization parameter together with the number of points in each grid. The sequential algorithm on Fig. 2 is successfully programmed in C. Its result shows that the computational time required takes a quite long time. It raises the latency issues that have implications for the arbitrage problems in asset pricing, as mentioned in the introduction. Afterward, this algorithm will be parallelized to reduce the latency of calculation process.

There are few important things to note related to the effective and efficient parallel algorithm design such as data dependence, load balance, concurrency and data synchronization [7]. Because of the independence of the data/instructions, only some parts of the algorithm that have a chance to be parallelized (parallel region). There are four parallel regions in the original C-GMM algorithm, i.e. STEP 1, STEP 3, outer-loop and inner-loop. Implementation results for the original C-GMM algorithm show that only the computation of outer-loop and inner-loop that requires a long time. Meanwhile, the other parallel regions, STEP 1 and STEP 3, only require the

![FIGURE 2. Determination of Optimal Parallel Region with and without Data Preprocessing](030146-4)
computational process in seconds where the experiments are related to the interval values and the number of points in the grid of each variable as well as the number of simulation. Furthermore, parallelization strategies for the parallel regions that are contribute significantly to the reduction of computational time will be discussed in the next two subsections.

**Parallel Region: The Outer-Loop at the Original C-GMM Algorithm**

The outer-loop at the original C-GMM algorithm calculate \( \sum_{TM}(\alpha_i, \theta_i, \nu) \) for each point in the grid for regularization parameter, \( \alpha_i \), which are evaluated. This results in a vector with size \( N \), i.e. \( \sum_{TM}(\bar{\alpha}, \theta_i, \nu) \). Suppose that there are \( p \) threads that can be used in distributing data/instructions. They have thread ID which starting from zero and ending at \( p-1 \). The outer-loop of the original C-GMM algorithm is parallelized by dividing the \( \bar{\alpha} \) into \( k^* \) independent subvectors where the \( k \)-th subvector with size \( N_k \), denoted by \( \bar{\alpha}_{N_k} = (\alpha_{i_k}, \alpha_{i_{k+1}}, \cdots, \alpha_{i_k}) \), for \( 0 \leq k \leq k^*-1 \leq p-1 \) and \( \sum_{k=0}^{k^*-1} N_k = N \). By denoting the other one subvector of \( \bar{\alpha} \) with size \( N_k^* \) as \( \bar{\alpha}_{N_k^*} \), then \( \alpha_{i_k} \neq \alpha_{i_{k^*}} \), for any element \( \alpha_{i_k} \) and \( \alpha_{i_{k^*}} \) in the two different random subvectors \( \bar{\alpha}_{N_k} \) and \( \bar{\alpha}_{N_k^*} \), respectively, for all \( 1 \leq i_k \leq N_k, 1 \leq s_{k^*} \leq N_k^* \), and \( 0 \leq k < k^* < k^*-1 \leq p-1 \). This parallelization strategy makes the calculation process of \( \sum_{TM}(\bar{\alpha}, \theta_i, \nu) \) being divided into \( k^* \) threads where \( k \)-th thread will process the calculation of a vector with size \( M, \theta_{TM}(\alpha_{i_k}) \), in the \( N_k \) iterations until they result also a vector with size \( N_k^* \), i.e. \( \sum_{TM}(\bar{\alpha}_{N_k}, \theta_i, \nu) \). Finally, all of the vectors will be joined into a vector with size \( N, \sum_{TM}(\bar{\alpha}, \theta_i, \nu) \). The flowchart in Fig. 3 illustrates parallelization strategy for the outer-loop of the original C-GMM algorithm where \( k^* = p \).

**FIGURE 3.** The Parallel Programming Flowchart for the Outer-Loop of the Original C-GMM Algorithm
Parallel Region: The Inner-Loop at the Original C-GMM Algorithm

Different with the outer-loop of the original C-GMM algorithm, the inner-loop optimize $\hat{\theta}_T^{(j)}(a_i)$ for each sample that was simulated in STEP 3, $X_T^{(j)}$, given $a_i$. This results in a matrix of size $M \times N$, i.e. $\hat{\theta}_T^{(m)}(a_i)$. In this case, the inner-loop at the original C-GMM algorithm is parallelized by dividing the simulated samples, $X_T^M$, into $k^*$ independent submatrices. Let the $k$-th submatrix with size $T \times M_k$ be given by:

$$X_T^{M_k} = \begin{pmatrix}
X_T^{(1k)} & X_T^{(2k)} & \cdots & X_T^{(M_k)} \\
X_T^{(1k)} & X_T^{(2k)} & \cdots & X_T^{(M_k)} \\
\vdots & \vdots & \ddots & \vdots \\
X_T^{(1k)} & X_T^{(2k)} & \cdots & X_T^{(M_k)} \\
\end{pmatrix}$$

where $0 \leq k \leq k^* - 1 \leq p - 1$ and $\sum_{k=0}^{k^*-1} M_k = M$ as illustrated in Fig. 4 for $k^* = p$. By denoting the other one submatrix of $X_T^M$ with size $T \times M_k$ as $X_T^{M_k^*}$ then $X_T^{(j_k)} \neq X_T^{(u_{k^*})}$ for any column vector $X_T^{(j_k)}$ and $X_T^{(u_{k^*})}$ in the two different random submatrices $X_T^{M_k}$ and $X_T^{M_k^*}$, respectively, for all $1 \leq j_k \leq M_k, 1 \leq u_{k^*} \leq M_k^*$ and $0 \leq k < k^* \leq k^* - 1 \leq p - 1$.

**FIGURE 4.** The Parallel Programming Flowchart for the Inner-Loop of the Original C-GMM Algorithm
The flowchart in Fig. 4 shows that the calculation process of $\hat{\theta}_{T,M_k}(\alpha_i)$ being divided into $k^{**}$ threads where $k$-th thread will process the calculation of a subvector of $\hat{\theta}_{T,M_k}(\alpha_i)$ with size $M_k$. This parallelization is programmed in $N$ iterations.

RESULTS AND DISCUSSION

In the Flynn's taxonomy, the parallelization of both parallel regions, the outer-loop and the inner-loop at the original C-GMM algorithm, are classified as Single Program Multiple Data (SPMD) [11] because of each thread will process the different data sets using same formula. In this paper, the parallel algorithm will be programmed with Open Multi-Processing (OpenMP) on the platform consists of a personal computer equipped with an Intel Core i7 Quad-Core at 2.3GHz and 8GB of RAM. OpenMP is a shared-memory application programming interface (API) which provides a directive compiler to facilitate shared-memory parallel programming [8]. This makes it easy for programmers by preventing a number of errors in programming through a structured and systematically approach. OpenMP can be written in three languages: C, C++ and Fortran where in this paper, the parallel algorithm will be programmed also in C with OpenMP 3.1. Parallelization strategies for the original C-GMM algorithm will be implemented to the Cox-Ingersoll-Ross (CIR) model that models short-rate at time $t$, denoted by $r(t)$, in the following stochastic differential equation [12]:

$$dr(t) = \kappa(\theta - r(t))dt + \sigma \sqrt{r(t)}dW(t), \quad (9)$$

with $\kappa, \theta, \sigma$ and $r(t)$ are non-negative and $W(t)$ is a Brownian motion under a risk-neutral probability measure $\bar{P}$ where:

$\kappa$ : speed of adjustment,
$\theta$ : reversion level of short-rate,
$\sigma^2r(t)$ : variance of short-rate.

It is seen that the drift coefficient depends on $r(t)$. If $r(t)$ is less than $\theta$, then the drift coefficient becomes positive, and vice versa. Therefore, short-rate moves toward a reversion level which depends on the speed of adjustment. Meanwhile, the diffusion coefficient describes that CIR model implicitly assumes short-rate as a non-negative stochastic process.

Suppose that $\hat{\theta} = (\kappa, \theta, \sigma)'$ to implement the C-GMM method in estimating the parameter of CIR model. Moreover, it can be also determined the conditional characteristic function of CIR model in equation (9), $r(t + \Delta)$, given $r(t)$, as follows [13]:

$$\psi_{r(t)}(\tau, \kappa, \theta, \sigma) = \left(1 - \frac{\tau}{\gamma} \right)^{\frac{\lambda}{\sigma^2}} \exp \left(\frac{-\kappa e^{-\lambda \Delta}}{1 - \gamma} r(t)\right), \quad (10)$$

FIGURE 5. The Speed-up of Parallel C-GMM Algorithm
where \( r(t) \) has a Poisson-mixing-Gamma distribution [14]. However, this paper limits the discussion for only finite values of MSE which is \( \nu = 0 \) and assumes that \( \Delta = 1 \) to simplify the problem. In this paper, it will be generated a sample of size \( T = 250 \) with a weekly sampling frequency in mind based on the true parameter which assumed same with the parameter estimates that obtained from Gallant and Tauchen [15], i.e. \( \kappa = 0.00285 \), \( \theta = 8.7403509 \) and \( \sigma = 0.0275 \). The performance of parallel algorithm for C-GMM will be analyzed in some subsets of this generated data series where the number of simulations and points in the grid for \( r \) and regularization parameter is same, i.e. 50 and 30, respectively, across all scenarios to keep the consistency of the results for each the number of observations that used in the parallel programming experiment. The speed-up is a measure for measuring the effectiveness of parallel algorithm that has been designed [16] which is defined as follows,

\[
S(n, p) = \frac{T(n, 1)}{T(n, p)},
\]

where \( T(n, p) \) is a computational time required for parallel the algorithm in processing a problem of size \( n \) with \( p \) threads \( p > 1 \) whereas \( T(n, 1) \) is denoted for sequential algorithm.

Figure 5 presents the speed-up of parallel the C-GMM algorithm: outer-loop, inner-loop and their combination, with 8 threads for all scenarios. It shows that on average, the outer-loop parallelization is the best strategy in this experiment, followed by the combination (outer-inner-loop) and the only inner-loop region. However, relatively, the larger the number of observations, the faster speed-up of parallel (inner-loop) C-GMM algorithm but opposite to two others. It might be caused by the optimization of \( \delta_T(\alpha) \) in the inner-loop region which is directly related to the number of observations. Different results might be obtained if the experiment is done by using more threads and different generated sample.

**CONCLUSIONS**

There are only two of four parallel regions in the original C-GMM algorithm, especially in optimizing regularization parameter: the outer-loop and the inner-loop, that are contributed significantly to the reduction of computational time. They are both classified as SPMD model. These parallelization strategies is effectively programmed under stochastic environment with standard shared-memory API, i.e. OpenMP 3.1 using directive compiler in C with 8 threads, that is implemented in the parameter estimation of Cox-Ingersoll-Ross model. Moreover, the outer-loop parallelization is the best strategy for any number of observations in this experiment. Parallel Algorithm for C-GMM can be developed on the Graphics Processing Unit (GPU) by using Compute Unified Device Architecture (CUDA) tool for implementing in the GPU card [17]. Moreover, parallelization can be also designed for developing C-GMM method itself, i.e. Indirect Continuous-Generalized Method of Moments (IC-GMM) [18].

**REFERENCES**