# Parallel Implementation of Monte Carlo-Markov Chain Algorithm

**Oscar A. Esquivel-Flores,**
**Óscar A. Alvarado-Morán**

## Table of contents

## Background

Solving systems of linear algebraic equations (SLAE) $Ax = b$ or inverting a real matrix $A$ are well-known and important problems.

Iterative solvers (IS) are used widely to compute SLAE solutions due to their predictability and reliability. Nevertheless they are prohibitive for large-scale problems

Monte Carlo Methods (MCM) are probabilistic methods, that use random numbers to simulate stochastic behaviour to estimate the solution of a problem.

MCM requires $O(NL)$ steps to find a single element of the solution. $N$ is the number of Markov chains and $L$ is an estimate of the chain length in the stochastic process

## Monte Carlo Methods

Consider the system $Ax = b$, after splitting $A = M - N$ the system could be transformed in $x_{k+1} = Tx_k + f$, $k = 0, 1, 2, \dots$ where $T = M^{-1}N$ and $f = M^{-1}b$, assuming $\|T\| < 1$ the iteration converges for any initial vector $X_0$

The Markov chain given by $\gamma : r_0 \to r_1 \to \cdots \to r_k \to \dots$ using random walks on the elements of $T$ in order to sample the solution of initial system.

Random walks are based on transition probabilities $P \in R^{n \times n}$ and $p \in R^n$.

For the stochastic trajectory $\gamma$ a unbiased estimator is $X(\gamma) = \sum_{m=0}^{\infty} W_m f_{r_m}$ where transition weight is defined as:

$$W_m = W_{m-1} w_{r_{m-1} r_m},\ w_{r_{m-1} r_m} = \frac{t_{r_{m-1} r_m}}{p_{r_{m-1} r_m}}$$

## Monte Carlo Algorithm

In practice is common to apply the partial sum $X(\gamma) = \sum_{m=0}^{k} W_m f_{r_m}$ on infinite Markov chain whose mathematical expectation tends to $x_i$ by choosing $k$ large enough.

To simulate $N$ independent sample paths of the Markov chain $\gamma : r_0^s \rightarrow r_1^s \rightarrow \cdots \rightarrow r_k^s \rightarrow \ldots$ for $s = 1, 2, \ldots, N$and considere the sample mean of $X_i(\gamma_k^s)$ to estimate the real mean $E[X_i(\gamma_k)]$.

Therefore, $\hat{X}_i(\gamma_k^s) = \frac{1}{N} \sum_{s=1}^{N} X_i(\gamma_k^s) \approx X_i$

## Monte Carlo Algorithm

In this way we encounter the statistical error, which is theoretically expressed as $\left|\overline{X}_i(\gamma_k) - E\left[X_i(\gamma_k)\right]\right| < \delta$, where $\delta$ is the given parameter.

From the practical point of view, probable error is employed to estimate the statistical error. By applying the Central Limit Theorem (CLT) for the random variable $\overline{X}_i(\gamma_k)$, we can obtain the probable error $r_N = 0.6745\sqrt{\frac{var(X_i(\gamma))}{N}}$.

Applying the precision $r_N \leq \delta$ and employing the upper bound for $var(X_i(\gamma))$, we obtain:

$$N \geq \frac{(0.6745)^2}{\delta^2} \frac{\|f\|^2}{(1 - \|f\|)^2}$$

## Parallel Algorithm

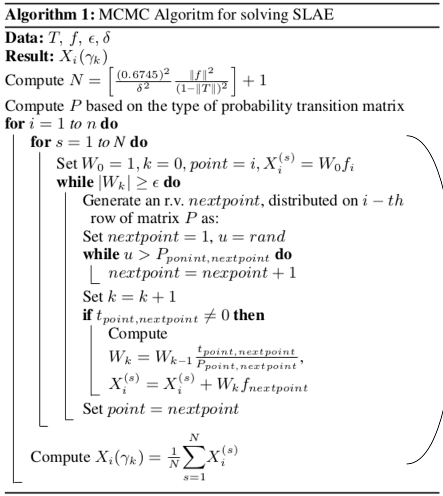
Recently parallel MCM was designed and developed with the following main generic properties:

Efficient distribution of the compute data

Minimum communication during the computation

Increased precision achieved by adding extra refinement computation

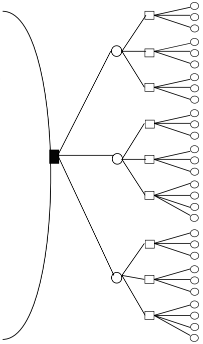
## Parallel Algorithm



**Algorithm 1:** MCMC Algoritm for solving SLAE

```
Data: $T,\ f,\ \epsilon, \delta$
Result: $X_i(\gamma_k)$
Compute $N = \left[\frac{(0.6745)^2}{\delta^2}\frac{\|f\|^2}{(1-\|T\|)^2}\right] + 1$
Compute $P$ based on the type of probability transition matrix
for $i = 1$ to $n$ do
    for $s = 1$ to $N$ do
        Set $W_0 = 1, k = 0, point = i, X_i^{(s)} = W_0 f_i$
        while $|W_k| \geq \epsilon$ do
            Generate an r.v. $nextpoint$, distributed on $i - th$
              row of matrix $P$ as:
            Set $nextpoint = 1,\ u = rand$
            while $u > P_{ponint,nextpoint}$ do
                $nextpoint = nexpoint + 1$
            Set $k = k + 1$
            if $t_{point,nextpoint} \neq 0$ then
                Compute
                $W_k = W_{k-1}\frac{t_{point,nextpoint}}{P_{point,nextpoint}}$,
                $X_i^{(s)} = X_i^{(s)} + W_k f_{nextpoint}$
            Set $point = nextpoint$
    Compute $X_i(\gamma_k) = \frac{1}{N}\sum_{s=1}^{N} X_i^{(s)}$
```

Parallel section



## Experiments

In order to studying the Marko Chain-Monte Carlo method for solving systems of linear algebraic equations it is relevant not only to take abstract matrices but matrices which represent a concrete problem to solve

Mathematically, the system $Ax = b$ is defined by the following partial equation

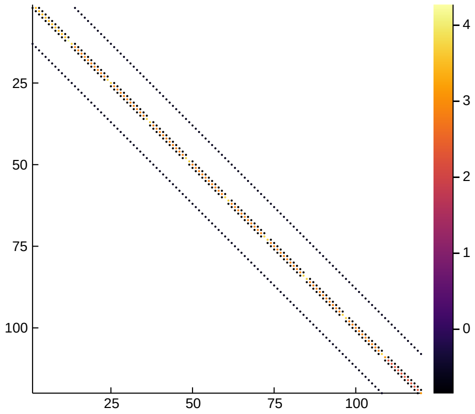
$$\frac{\partial(\rho C_v T)}{\partial t} = \nabla \cdot (\kappa \nabla T)$$

where $\rho$, $t$, $C_v$, $T$ and $\kappa$ represent density, time, specific heat, temperature and thermal conductivity respectively.

It's worth to mention that test matrix was obtained with finite volume method; dimensions of test matrix $A$ are of $120 \times 120$ and vector $b$ is a column vector of size 120.

# Experiments

Matrix $A$ is used as input to the Monte Carlo - Markov Chain Algorithm to compute the accuracy of the approximate solution results of MCMC process variiying the $\epsilon$ and $\delta$ parameters. Figure shows the sparsity pattern of the test matrix.



## Experiments

Table hows the compute times and the 2-norms of $\|b - A\hat{x}\|_2$ for different values of parameters. Number of Markov Chains $N$, $\|b - A\hat{x}\|_2$ and execution time (sequential and parallel) for different values of $\epsilon$ and $\delta$

| $\epsilon$ | $\delta$ | $N$ | $\|\cdot\|_2$ | sectime(s) | partime(s) |
|---|---|---|---|---|---|
| 0.1 | 0.1 | 4 | 6.729 | 0.188 | 0.044 |
| 0.1 | 0.05 | 14 | 5.174 | 0.553 | 0.130 |
| 0.1 | 0.01 | 330 | 1.05 | 12.07 | 2.68 |
| 0.1 | 0.005 | 1319 | 0.56 | 47.62 | 10.58 |
| 0.1 | 0.001 | 32961 | 0.22 | 1203.2 | 261.52 |
| 0.05 | 0.1 | 4 | 6.69 | 0.263 | 0.0438 |
| 0.05 | 0.05 | 14 | 5.67 | 0.662 | 0.118 |
| 0.05 | 0.01 | 330 | 0.94 | 14.168 | 2.28 |
| 0.05 | 0.005 | 1319 | 0.50 | 56.96 | 17.24 |
| 0.05 | 0.001 | 32961 | 0.154 | 1435.9 | 224.21 |
| 0.01 | 0.1 | 4 | 6.65 | 0.292 | 0.047 |
| 0.01 | 0.05 | 14 | 3.60 | 0.839 | 0.144 |
| 0.01 | 0.01 | 330 | 1.06 | 19.12 | 3.78 |
| 0.01 | 0.005 | 1319 | 0.55 | 76.25 | 11.91 |
| 0.01 | 0.001 | 32961 | 0.154 | 2142.0 | 310.88 |
| 0.005 | 0.1 | 4 | 6.92 | 0.323 | 0.052 |
| 0.005 | 0.05 | 14 | 3.21 | 0.948 | 0.148 |
| 0.005 | 0.01 | 330 | 1.05 | 20.79 | 3.19 |
| 0.005 | 0.005 | 1319 | 0.69 | 84.62 | 12.48 |
| 0.005 | 0.001 | 32961 | 0.11 | 2149.7 | 338.53 |

## Conclusions

MCM is a good option as solver and preconditioner in parallel, taking advantage of inaccurate approximations as result.

Analyse and design parallel algorithms are not easy tasks, however **Julia 1.4.0** help us to wrap several GPU'S functionalities through **CUDA.jl**.

**Random Numbers**

Some problems about computation of random numbers into the kernel become in loss of accuracy.

## Thanks...

To Tim Besard and all generous (and very clever) people who work hard to translate CUDA into Julia.

To all Julia Language Team and friends for creating this cute, attractive and powerful programming language and save JuliaCon 2020.

## References I

Behrouz Fathi-Vajargah and Zeinab Hassanzadeh.
Improvements on the hybrid monte carlo algorithms for matrix computations.
*Sādhanā*, 44(1):1, Dec 2018.

V. Alexandrov et al.
*Parallel Hybrid Monte Carlo Algorithms for Matrix Computations*
Computational Science – ICCS 2005
Springer Berlin / Heidelberg, 2005.

V. Alexandrov, O. Esquivel-Flores
*Towards Monte Carlo Preconditioning Approach and Hybrid Monte Carlo Algorithms for Matrix Computations*
Computers and Mathematics with Applications, 2015

G.H. Golub and C. Van Loan
*Matrix computations*

## References II

A. Lebedev and V. Alexandrov.
*On advanced monte carlo methods for linear algebra on advanced accelerator architectures*
2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA), pages 81–90, Nov 2018.

Tim Besard, Christophe Foket, and Bjorn De Sutter.
*Effective extensible programming: Unleashing julia on gpus*.
arXiv:1712.03112 2017

Oscar A. Esquivel-Flores,
Óscar A. Alvarado-Morán

# Parallel Implementation of Monte Carlo-Markov Chain Algorithm