Egyptian Informatics Journal (2015) 16, 37–44

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGINAL ARTICLE

Cluster computing for the large scale discrete fractional Cable equation

N.H. Sweilam [*], Hatem Moharram, N.K. Abdel Moniem

Mathematics Department, Faculty of Science, Cairo University, Egypt

Received 17 July 2014; revised 17 December 2014; accepted 20 December 2014
Available online 29 March 2015

KEYWORDS

Weighted average finite difference method;
Fractional Cable equation;
Precondition conjugate gradient method (PCG);
Parallel computations;
Linux PC cluster workstation

Abstract This paper presents a numerical simulation technique for the fractional Cable equation in large scale domain. Special attention is given to the parallel execution of the fractional weighted average finite difference method (FWA-FDM) on distributed system with explicit message passing, where the fractional derivative is defined in Riemann–Liouville sense. The resultant huge system of equations is studied using precondition conjugate gradient method (PCG), with the implementation of cluster computing on it. The proposed approach fulfills the suitability for the implementation on Linux PC cluster through the minimization of inter-process communication. To examine the efficiency and accuracy of the proposed method, numerical test experiments using different number of the Linux PC cluster nodes are studied. The performance metrics clearly show the benefit of using the proposed approach on the Linux PC cluster in terms of execution time reduction and speedup with respect to the sequential running in a single PC.

1. Introduction

It is known from medicine that a nerve cell is an electrically excitable cell that processes and transmits information through electrochemical signals. These signals between nerve cells occur via synapses, specialized connections with other cells. The brain is an organ that serves as the center of the nervous system. The number of nerve cells in the brain varies dramatically from species to species. One estimate puts the human brain at about 100 billion nerve cells and 100 trillion synapses. The function of the brain is to exert centralized control over the other organs of the body. Diffusion plays a crucial role in brain function. The fractional Cable equation plays an important role to model electrodiffusion of ions in nerve cells with anomalous subdiffusion along and across the nerve cells.

PCs cluster system is one of low-cost general-purpose parallel computing systems. Cluster computing is currently one of the most successful alternatives for dealing with these challenging problems, which usually require high computation power. The well-known Message Passing Interface (MPI) is of the few representatives of the parallel programming paradigm for clusters. However, a number of problematic tasks arise from its implementation in the cluster context, in particular,

[*] Corresponding author.
E-mail address: nsweilam@sci.cu.edu.eg (N.H. Sweilam).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.eij.2014.12.001
1110-8665 © 2015 Production and hosting by Elsevier B.V. on behalf of Faculty of Computers and Information, Cairo University.

devising new high performance computing algorithms using the MPI and suitable for the cluster infrastructures and their effective mapping, scheduling and cross-nodes execution.

In recent years a wide variety of biological systems have shown anomalous diffusion, and its rates cannot be characterized by a single parameter of the diffusion constant [1]. Anomalous diffusion in these biological systems deviates from the standard Fichean description of Brownian motion, the main character of which is that its mean squared displacement is a nonlinear growth with respect to time, such as $x^2\ (t) \sim t^{\infty}$. As examples, single particle tracking experiments have revealed subdiffusion ($0 < \alpha < 1$) of proteins and lipids in a variety of cell membranes [1–4]. Anomalous subdiffusion has also been observed in neural cell adhesion molecures [5]. Indeed, anomalous diffusion occurs in many other physical situations, such as, transport of fluid in porous media [6], and the propagation of mechanical diffusive waves in viscoelastic media [7].

Due to its significant deviation from the dynamics of Brownian motion, the above-mentioned anomalous diffusion in biological systems cannot be adequately described by the traditional Nernst–Planck equation or its simplification, the Cable equation. Very recently, a modified Cable equation was introduced for modeling the anomalous diffusion in spiny neuronal dendrites [8]. The resulting governing equation, the so-called fractional Cable equation, is similar to the traditional Cable equation except that the order of derivative with respect to the space and/or time is fractional.

The main aim of this paper is to simulate the numerical solutions of the fractional Cable equations in large scale domain. The parallel FWA-FDM is used on distributed system with explicit message passing.

In this section, the definitions of the Riemann–Liouville and the Grünwald–Letnikov fractional derivatives are given as follows [9]:

Definition 1. The Riemann–Liouville derivative of order α of the function $y(x)$ is defined by:

$$D_x^{\alpha} y(x) = \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dx^n} \int_0^x \frac{y(\tau)}{(x-\tau^{\alpha-n+1})} d\tau, \quad x > 0 \tag{1}$$

where n is the smallest integer exceeding α and $\Gamma(.)$ is the Gamma function. If $\alpha = m \in N$, then (1) coincides with the classical mth derivative $y^{(m)}(x)$.

Definition 2. The Grünwald–Letnikov definition for the fractional derivatives of order $\alpha > 0$ of the function $y(x)$ is defined by:

$$D^{\alpha} y(x) = \lim_{h \to 0} \frac{1}{h^{\alpha}} \sum_{k=0}^{[\frac{x}{h}]} w_k^{(\alpha)} y(x - hk), \quad x \geq 0, \tag{2}$$

where $[\frac{x}{h}]$ means the integer part of $\frac{x}{h}$ and $w_k^{(\alpha)}$ is the normalized Grünwald weights which are defined by $w_k^{(\alpha)} = (-1)^k \binom{\alpha}{k}$.

The Grünwald–Letnikov definition is simply a generalization of the ordinary discretization formula for integer order derivatives. The Riemann–Liouville and the Grünwald–Letnikov approaches coincide under relatively weak conditions; if $y(x)$ is continuous and $y'(x)$ is integrable in the interval [0, x], then for every order $0 < \alpha < 1$ both the Riemann–Liouville and the Grünwald–Letnikov derivatives exist and coincide for any value inside the interval This fact of fractional calculus ensures the consistency of both definitions for most physical applications, where the functions are expected to be sufficiently smooth [9,10].

The plan of the paper is as follows: In the second section some discrete versions of the fractional derivatives are given. Also, FWA-FDM is introduced. In the third section the PCG method is presented to study the model problem. In the fourth section the resulting tri-diagonal system from discretization of the Cable equation is solved by a parallel PCG method using MPI, some test examples are presented. The paper ends with some conclusions in the fifth section.

2. Approximate formula for fractional derivative

Let us consider the initial-boundary value problem of the fractional Cable equation which is usually written in the following way (see [11–15,20] and the reference cited therein):

$$u_t(x,t) = D_t^{1-\beta} u_{xx}(x,t) - \mu D_t^{1-\alpha} u(x,t), \quad a < x < b, \quad 0 \prec t \leq T \tag{3}$$

where $0 < \alpha, \beta \leq 1$, μ is a constant and $D_t^{1-\gamma}$ is the fractional derivative defined by the Riemann–Liouville operator of order $1-\gamma$, where $\gamma = \alpha, \beta$. Under the zero boundary conditions

$$u(a,t) = u(b,t) = 0, \tag{4}$$

and the following initial condition:

$$u(x,0) = g(x). \tag{5}$$

2.1. Finite difference scheme for the fractional Cable equation

In this section, we will use the FWA-FDM to obtain the discretization finite difference formula of the Cable Eq. (3). We use the following notations: Δt and Δx, as the time-step length and the space-step length, respectively. The coordinates of the mesh points are $x_j = a + j\Delta x$ and $t_m = m\Delta t$, and the values of the approximate solution $u(x,t)$ on these grid points are $u(x_j, t_m) \equiv u_j^m \approx U_j^m$. For more details about discretization in fractional calculus see [16,17].

In the first step, the ordinary differential operators are discretized as follows [16]:

$$\left.\frac{\partial u}{\partial t}\right|_{x_j, t_m + \frac{\Delta t}{2}} = \delta_t u_j^{m+\frac{1}{2}} + O(\Delta t) \equiv \frac{u_j^{m+1} - u_j^m}{\Delta t} + O(\Delta t), \tag{6}$$

and

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_j, t_m} = \delta_{xx} u_j^m + O(\Delta x)^2 \equiv \frac{u_{j-1}^m - 2u_j^m + u_{j+1}^m}{(\Delta x)^2} + O(\Delta x)^2. \tag{7}$$

In the second step, the Riemann–Liouville operator is discretized as follows:

$$D_t^{1-\gamma} u(x,t)|_{x_j, t_m} = \delta_t^{1-\gamma} u_j^m + O\ ((\Delta t)^p), \tag{8}$$

where

$$\delta_t^{1-\gamma}u_j^m \equiv \frac{1}{(\Delta t)^{1-\gamma}}\sum_{k=0}^{\left[\frac{t_m}{\Delta t}\right]} w_k^{(1-\gamma)}u(x_j,t_m-k\Delta t)$$
$$= \frac{1}{(\Delta t)^{1-\gamma}}\sum_{k=0}^{m} w_k^{(1-\gamma)}u_j^{m-k}, \tag{9}$$

where $\left[\frac{t_m}{\Delta t}\right]$ means the integer part of $\frac{t_m}{\Delta t}$ and for simplicity we choose $h = \Delta t$. There are many choices of the weights $w_k^{(\alpha)}$ [16,18], so the above formula is not unique. Let us denote the generating function of the weights $w_k^{(\alpha)}$ by $w(z,\alpha)$, i.e.,

$$w(z,\alpha) = \sum_{k=0}^{\infty} w_k^{(\alpha)}z^k. \tag{10}$$

if

$$w(z,\alpha) = (1-z)^{\alpha}, \tag{11}$$

then (9) gives the backward difference formula of the first order, which is called the Grünwald–Letnikov formula. The coefficients $w_k^{(\alpha)}$ can be evaluated by the recursive formula

$$w_k^{(\alpha)} = \left(1-\frac{\alpha+1}{k}\right)w_{k-1}^{(\alpha)}, \quad w_0^{(\alpha)} = 1. \tag{12}$$

For $\gamma = 1$ the operator $D_t^{1-\gamma}$ becomes the identity operator so that, the consistency of Eqs. (8) and (9) requires $w_0^{(0)} = 1$, and $w_k^{(0)} = 0$ for $k \geqslant 1$, which in turn means that $w(z,0) = 1$.

Now, to obtain the finite difference scheme of the Cable Eq. (3), we evaluate this Equation at the intermediate point of the grid (x_j, t_m):

$$\left[u_t(x,t) - D_t^{1-\beta}u_{xx}(x,t)\right]_{x_j,t_m} + \mu D_t^{1-\alpha}u(x_j,t_m) = 0. \tag{13}$$

Then, we replace the first order time-derivative by the forward difference formula (6) and replace the second order space-derivative by the weighted average of the three-point centered formula (7) at the times t_m and t_{m+1}

$$\delta_t u_j^m - \left\{\lambda\delta_t^{1-\beta}\delta_{xx}u_j^m + (1-\lambda)\delta_t^{1-\beta}\delta_{xx}u_j^{m+1}\right\} + \mu\delta_t^{1-\alpha}u_j^m = T_j^m, \tag{14}$$

with λ being the weight factor and T_j^m is the resulting truncation error. The standard difference formula is given by

$$\delta_t U_j^m - \{\lambda\delta_t^{1-\beta}\delta_{xx}U_j^m + (1-\lambda)\delta_t^{1-\beta}\delta_{xx}U_j^{m+1}\} + \mu\delta_t^{1-\alpha}U_j^m = 0. \tag{15}$$

Now, by substituting from the difference operators given by (6), (7) and (9), we get

$$\frac{U_j^{m+1}-U_j^m}{\Delta t} - \lambda\frac{1}{(\Delta t)^{1-\beta}}\sum_{r=0}^{m} w_r^{(1-\beta)}\left(\frac{U_{j-1}^{m-r}-2U_j^{m-r}+U_{j+1}^{m-r}}{(\Delta x)^2}\right)$$
$$-(1-\lambda)\frac{1}{(\Delta t)^{1-\beta}} \times \sum_{r=0}^{m} w_r^{(1-\beta)}\left(\frac{U_{j-1}^{m+1-r}-2U_j^{m+1-r}+U_{j+1}^{m+1-r}}{(\Delta x)^2}\right)$$
$$+\mu\frac{1}{(\Delta t)^{1-\alpha}}\sum_{r=0}^{m} w_r^{(1-\alpha)}U_j^{m-r} = 0. \tag{16}$$

Put $N_\beta = \frac{(\Delta t)^\beta}{(\Delta x)^2}$, $N_\alpha = (\Delta t)^\alpha$, $\varphi = (1-\lambda)N_\beta$, and under some simplifications we can obtain the following form:

$$-\varphi U_{j-1}^{m+1} + (1+2\varphi)U_j^{m+1} - \varphi U_{j+1}^{m+1} = R, \tag{17}$$

where

$$R = U_j^m + N_\beta\sum_{r=0}^{m}\left[\lambda w_r^{(1-\beta)} + (1-\lambda)w_{r+1}^{(1-\beta)}\right]$$
$$\left[U_{j-1}^{m-r} - 2U_j^{m-r} + U_{j+1}^{m-r}\right] - \mu N_\alpha\sum_{r=0}^{m} w_r^{(1-\alpha)}U_j^{m-r}. \tag{18}$$

Equation (17) can be transferred into a matrix form as follows:

$$A = \begin{bmatrix} a_1^{(m)} & d_1^{(m)} & 0 & \cdots & \cdots \\ d_2^{(m)} & a_2^{(m)} & 0 & 0 & 0 \\ & \ddots & \ddots & \ddots & 0 \\ & & d_{N-2}^{(m)} & a_{N-2}^{(m)} & d_{N-2}^{(m)} \\ & & & d_{N-1}^{(m)} & a_{N-1}^{(m)} \\ & & & & \end{bmatrix}$$

where $b_m = \left[b_1^{(m)}, b_2^{(m)}, \ldots, b_{N-2}^{(m)}, b_{N-1}^{(m)}\right]$,

$u_m = \left[u_1^{(m)}, u_2^{(m)}, \ldots, u_{N-2}^{(m)}, u_{N-1}^{(m)}\right], a_j^{(k)} = 1+2\phi, j$

$= 1,2,\ldots,m-1, m = 1,2,\ldots,M-1, d_j^{(k)} = \phi.$

The fractional weighted average difference scheme is Eq. (17). Fortunately, Eq. (17) is tridiagonal system that can be solved using conjugate gradient method. In the case of $\lambda = 1$ and $\lambda = \frac{1}{2}$ we have the backward Euler fractional quadrature method and the Crank–Nicholson fractional quadrature methods, respectively, which have been studied e.g., in [19], but at $\lambda = 0$ the scheme is called fully implicit. For the stability analysis see [20].

2.2. The PCG method

The aim now is to introduce Eq. (17), at each time step, to solve a triangular system of linear equations where the right-hand side utilizes all the history of the computed solution up to that time. We use the PCG method to solve this system.

The primary introduction of the conjugate gradient algorithm from 1952 can be found in [21]. This method is used to solve linear systems of the form:

$$Ax = b \tag{19}$$

With (symmetric, positive definite) and b $\in R^n$. In this case the solution of the linear system is equivalent to the minimum function [22]:

$$E: R^n \to R, x \to \frac{1}{2}x^TAx - x^Tb$$

Meaning x solves $Ax = b$ if and only if E has a global minimum at x.

This equivalence is the basic idea of the conjugate gradient algorithm. Instead of solving a linear system in a typical way, we search the minimum of the function E. Let $x = x_0 \in R^n$ be an arbitrary start vector. We search the minimum of E on the line

$$g: R^n \to R, \alpha = x + \alpha p.$$

To obtain the minimum approximately we use an iterative search with different search directions.

The conjugate gradient method works very well on matrices that are well-conditioned (i.e. condition number is not too big); however, in real applications, most matrices are ill-conditioned (i.e. the condition number is large), reducing the efficiency of the algorithm. To increase efficiency of the algorithm we use Preconditioning technique for improving the condition number of a matrix. By using precondition [23] we can iteratively solve Eq. (19) more quickly than the original problem. The idea behind preconditioning is to use the CG method on an equivalent system. Thus, instead of solving $Ax = b$ we solve a related problem $\widetilde{A}\tilde{x} = \tilde{b}$, for which $\widetilde{A}$ is chosen such that its condition number is closer to one; in other words, $\widetilde{A}$ is close to the identity.

2.3. Mathematical formulation for PCG method

Let $\widetilde{A}\tilde{x} = \tilde{b}$ be the transformed system of $\widetilde{A}\tilde{x} = \tilde{b}$. The two systems are related through the following relationships [24]:

$$\widetilde{A} = B^{1/2}A, x = B^{1/2}y, \tilde{b} = b^{1/2}y$$

where we picked $B^{1/2}$ to be a symmetric positive-definite matrix and $y = B^{-1/2}x$. B is called a preconditioner. Making a change of notation, let $C = B^{1/2}AB^{1/2}$. Then, instead of solving $Ax = b$, we have to solve the following related problem:

$$\begin{aligned} &Cy = B^{1/2}b, \\ &x = B^{1/2}y. \end{aligned} \tag{20}$$

The simplest preconditioner is a diagonal matrix whose diagonal entries are identical to those of in this paper, we apply this preconditioner, known as the diagonal preconditioning or the Jacobi preconditioning [25]. The PCG algorithm has the following two parts, which are repeatedly executed until the convergence of PCG method is performed by checking the error criteria i.e. Euclidean norms of the residual vector should be less than prescribed tolerance.

2.3.1. Part (a): compute the new iterate $\mathbf{y}_{k+1}$, the search parameter α_k, and the residual $\mathbf{r}_{k+1}$

In order to calculate y, we set an arbitrary start vector x and calculate a more precise approximation of the minimum in every iteration

$y_{k+1} \leftarrow y_k + \alpha_k p_k$.

If we replace y_k with $B^{-1/2}x_k$. while calculating value of y_{k+1}, then the equation becomes:

$$B^{-1/2}x_{k+1} \leftarrow B^{-1/2}x_k + \tilde{\alpha}_k\tilde{p}_k$$

Multiplying both sides by $B^{-1/2}$ from the left, value of y_{k+1} is further transformed into

$$x_{k+1} \leftarrow x_k + \tilde{\alpha}_k B^{1/2}\tilde{p}_k, \text{ or}$$

$$x_{k+1} \leftarrow x_k + \tilde{\alpha}_k\tilde{p}_k$$

We can find an analytic formula for α_k. For fixed y_k and p_k,

$$\begin{aligned} f(y_k + \alpha_k p_k) &= \tfrac{1}{2}(y_k + \alpha_k p_k)^T C(y_k + \alpha_k p_k) - (y_k + \alpha_k p_k)^T b \\ &= \tfrac{1}{2}\alpha^2 p_k^T C p_k + \alpha p_k^T C y_k + -\alpha p_k^T b + \ldots \end{aligned}$$

The minimum of f with respect to α occurs when the derivative is zero:

$$p_k^T C y_k + \alpha p_k^T C p_k + -p_k^T b = 0$$

so

$$\alpha_{k+1} = \frac{p_k^T(Cy_k - b)}{p_k^T C p_k} = \frac{p_k^T r_k}{p_k^T C p_k} = \frac{r_k^T r_k}{p_k^T C p_k}.$$

for proof of equivalence see [24],

The residuals $r_{k+1} = b - Cy_{k+1}$ can be computed iteratively using

$$r_{k+1} \leftarrow r_k - \alpha_{k+1} C p_k (\text{residual})$$

because

$$r_k - \alpha_{k+1} C p_k = b - C(y_k + \alpha p_k) = b - Cy_{k+1} = r_{k+1}.$$

A relationship between the initial residual $\tilde{r}_0$ of Eq.(20) and the initial residual r_0 of $Ax = b$, can be found by the following:

$B^{1/2}b - Cy_0 = \tilde{r}_0$, multiplying both sides of $B^{1/2}b - B^{1/2}AB^{1/2}y_0 = \tilde{r}_0$ by $B^{-1/2}$, form left, we obtain $b - A(B^{1/2}y_0) = B^{-1/2}\tilde{r}_0$ or $b - Ax_0 = B^{-1/2}\tilde{r}_0$. Thus $r_0 = B^{-1/2}\tilde{r}_0$.

Generalizing we obtain $r_k = B^{-1/2}\tilde{r}_k$.

Similarly we can obtain $p_k = B^{-1/2}\tilde{p}_k$.

In all the computations to follow, we will substitute $B^{-1/2}\tilde{r}_k$ with r$_k$. By replacing $\tilde{r}_k$ by $B^{1/2}r_k$ the residual equation becomes:

$$\begin{aligned} B^{1/2}r_{k+1} &= B^{1/2}r_k - \tilde{\alpha}_{k+1}C\tilde{p}_k \text{ or } B^{1/2}r_{k+1} \\ &= B^{1/2}r_k - \tilde{\alpha}_{k+1}B^{1/2}\mathrm{A}B^{1/2}\tilde{p}_k \end{aligned}$$

multiplying both sides, from left, by $B^{-1/2}$ we get:

$$r_{k+1} = r_k - \tilde{\alpha}_{k+1}\mathrm{A}B^{1/2}\tilde{p}_k, \text{ or } r_{k+1} = r_k - \tilde{\alpha}_{k+1}\mathrm{A}p_k$$

2.3.2. Part (b): Compute the new search direction

Since C is positive definite and therefore $r_{k+1}Cr_k \neq 0$. The search directions can be created iteratively. With the approach

$p_{k+1} = r_{k+1} + \beta_{k+1}p_k$ and $p_0 = r_0$.(search direction)

multiplying both sides, from left, by $B^{1/2}$ we get:

$$B^{1/2}\tilde{p}_{k+1} = B^{1/2}\tilde{r}_{k+1} + \tilde{\beta}_{k+1}B^{1/2}\tilde{p}_k \text{ i.e. } p_{k+1} = B^{1/2}\tilde{r}_{k+1} + \tilde{\beta}_{k+1}p_k$$
$$\text{or } p_{k+1} = B^{1/2}(B^{1/2}r_{k+1}) + \tilde{\beta}_{k+1}p_k \text{ or } p_{k+1} = Br_{k+1} + \tilde{\beta}_{k+1}p_k$$

We also find a new formula for the improvement $\tilde{\beta}_{k+1}$

$$\tilde{\beta}_{k+1} = \frac{(B^{1/2}r_{k+1})^T(B^{1/2}r_{k+1})}{(B^{1/2}r_k)^T(B^{1/2}r_k)} = \frac{r_{k+1}^T B^{1/2}B^{1/2}r_{k+1}}{r_k^T B^{1/2}B^{1/2}r_k},$$
$$\text{or } \tilde{\beta}_{k+1} = \frac{r_{k+1}^T B r_{k+1}}{r_k^T B r_k}.$$

2.4. The iterative formulas of PCG are given below

$x_{k+1} \leftarrow x_k + \tilde{\alpha}_k\tilde{p}_k$.	// Approximate solution at step k + 1
$r_{k+1} \leftarrow r_k - \alpha_{k+1}Cp_k$.	// Residual
$p_{k+1} = Br_{k+1} + \tilde{\beta}_{k+1}p_k$.	// search direction:
$\tilde{\beta}_{k+1} = \frac{r_{k+1}^T Br_{k+1}}{r_k^T Br_k}$.	// improvement at step k
$\alpha_{k+1} = \frac{r_k^T r_k}{p_k^T C p_k}$.	//step length:

2.5. Parallel implementation with row-wise block-striped [26]

The parallel algorithm is the same as the serial but some exceptions the parallel implementation have which are

1. Data reading from the input_le and dividing it across processors (using MPI_Bcast and MPI_Scatter).
2. After each processor computes inner product locally, sum reduction across all processors is required (using MPI_Allreduce).
3. The vector matrix product requires gathering all the local parts of the vector into a single vector then each processor do the multiplication(using MPI_Allgather).

The following is the parallel code fragment for the PCG for solving linear systems of the form $Ax = b$.

```
r_local = b_local                    // initial residual
rho = Allreduce (r_local' * r_local)        // an intermediate term
                                           used later.
for k = 1:itermax
  if k = 1
    p_local = r_local              // initial direction.
  else
    beta = rho/oldrho              // improvement at step k i.e. $\beta$
    p_local = r_local + beta* p_local   // search direction
  end
  p = Gather(p_local)    // sum of search direction over all
processors
  v_local = C_local * p   // intermediate term to be used
  alpha = rho/ Allreduce(p_local' * v_local)   // value of $\alpha$ all over
                                               all processors.
  x_local = x_local + alpha * p_local       // solution vector
  r_local = r_local − alpha * v_local       // new residual
  oldrho = rho          // intermediate term needed to be used as
term
                        // from previous step.
  rho = Allreduce (r_local' * r_local)     // new intermediate term
                                           used in current step
end
```

3. Stability analysis and truncation error

Theorem 3.1. The truncation error of Eq. (3) is given by

$$T_j^m = O(h^p) + (\frac{1}{2} - \lambda)O(\Delta t) + O(\Delta t)^2 + O(\Delta x)^2 + \frac{1}{h^{1-\beta}} w_{m+1}^{(1-\beta)} \delta_{xx} u_j^{(0)}.$$

Proof. From the definition of truncating error given by Eq. (14), one gets:

$$T_j^m = \delta_t u_j^m - \{\lambda \delta_{tt}^{1-\beta} \delta_{xx} u_j^m + (1-\lambda)\delta_{tt}^{1-\beta}\delta_{xx} u_j^{m+1}\},$$

i.e.

$$T_j^m = \delta_t u_j^m - \frac{1}{h^{1-\beta}} \sum_{r=0}^{m} w_r^{(1-\beta)} \left[(1-\lambda)\delta_{xx} u_j^{m+1-r} + \lambda \delta_{xx} u_j^{m-r}\right] - \frac{1}{h^{1-\beta}}(1-\lambda) w_{m+1}^{(1-\beta)} \delta_{xx} u_j^{(0)}. \quad (21)$$

But

$$\delta_{xx} u_j^{m+1-r} = u_{xx} + \frac{(\Delta x)^2}{12} u_{xxxx} + \frac{\Delta t}{2}\left[u_{xxt} + \frac{(\Delta x)^2}{12} u_{xxxxt} + \ldots\right] + \frac{(\Delta t)^2}{8} u_{xxtt} + \ldots,$$

And

$$\delta_{xx} u_j^{m+1-r} = u_{xx} + \frac{(\Delta x)^2}{12} u_{xxxx} + \frac{\Delta t}{2}\left[u_{xxt} + \frac{(\Delta x)^2}{12} u_{xxxxt} + \ldots\right] + \frac{(\Delta t)^2}{8} u_{xxtt} + \ldots,$$

where the partial derivatives are evaluated at the point $(x_j, t_{m-k} + \frac{\Delta t}{2})$. Inserting these expressions into Eq. (21) and taking into account Eqs. (3) and (9), we can get:

$$T_j^m = O(h^p) - \left(\frac{1}{2} - \lambda\right)\Delta t D_\tau^{1-\beta} u_{xxt} - \frac{(\Delta x)^2}{12} D_\tau^{1-\beta} u_{xxxx} - \frac{(\Delta t)^2}{8} D_\tau^{1-\beta} u_{xxtt} - \frac{1}{h^{1-\beta}}(1-\lambda) w_{m+1}^{(1-\beta)} \delta_{xx} u_j^{(0)} + \ldots,$$

With $\tau = t_m$ i.e.

$$T_j^m = O(h^p) + \left(\frac{1}{2} - \lambda\right) O(\Delta t) + O(\Delta t)^2 + O(\Delta x)^2 + \frac{1}{h^{1-\beta}} w_{m+1}^{(1-\beta)} \delta_{xx} u_j^{(0)}$$

where terms of order $O[(\Delta t)^a (\Delta x)^b h^p]$ with $a + b + p > 2$ have not been included.□

4. Performance analysis and PC cluster description

The main point of parallel computing is to run computations faster. Faster obviously means "in less time," but we immediately wonder, "How much less?" To understand both what is possible and what we can expect to achieve, we use several metrics to measure parallel performance, each with its own strengths and weaknesses. The first is the execution time. The execution time T_P, refers to the net execution time of a parallel program on P processors exclusive of initial OS, I/O, etc. charges. The second is the Speedup. Speedup S_P, is the execution time of a sequential program divided by the execution time of a parallel program that computes the same result. In particular, $S_P = T_S/T_P$, where T_S is the sequential execution time and T_P is the parallel execution time on P processors. The third is the efficiency. Efficiency E_P, is a normalized measure of speedup: $E_P = S_P/P$. Ideally, speedup should scale linearly with P, implying that efficiency should have a constant value of 1. Of course, because of various sources of performance loss, efficiency is more typically below 1, and it diminishes as we increase the number of processors. Efficiency greater than 1 represents superlinear speedup [26].

The parallel PCG has been implemented on the computer cluster of Faculty of Science Cairo University [27], a cluster of 17 workstations, one master and eight slaves, similar to Fig. 1. Each slave in the cluster has a microprocessor Intel(R) core(TM)2 i7-2600@3,40 GHz and 8 Gb DDR3 RAM. The master has a microprocessor Intel(R) core(TM)2 Quad CPU Q6700 2.66 GHz and 4 Gb DDR2 RAM. The slaves are

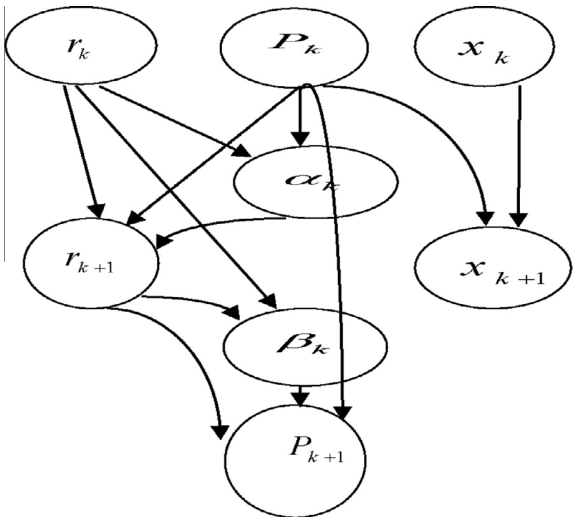

Figure 1 Data dependency of PCG algorithm.

connected to the master via star local network (1 Gbps). The network consists of a switch, a network card in each workstation and the corresponding Cat.6 UTP network wires (see Fig. 2).

The operating system of the cluster is Linux which includes tools for controlling the execution of parallel applications. As programming language C++ is selected due to its ease to manage large arrays of typical lattice model data combined with the Message Passing Interface (MPI) libraries.

5. Experimental results

In the following section we choose two examples which we have the exact solution of them so we can check the correctness of our results.

In Fig. 3 the behavior of the approximate solution with $\Delta x = \frac{1}{4000}, \Delta t = \frac{1}{40}$ *at* $\alpha = 0.2, \beta = 0.7$ is shown in a 3-D figure to display the simulation of fractional Cable equation in 3-D. While in Fig. 3 we compare the exact solution with the numerical solution.

Example 1. Consider the following initial-boundary problem of the fractional Cable equation

$$u_t(x,t) = D_t^{1-\beta}u_{xx}(x,t) - D_t^{1-\alpha}u(x,t) + f(x,t), \tag{22}$$

on a finite domain $0 < x < 1$, with $0 \leqslant t \leqslant T$, $0 < \alpha, \beta < 1$ and the following source term:

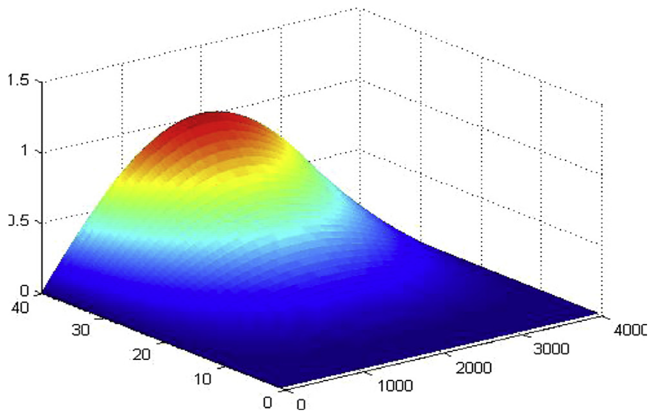

Figure 3 Approximate solution with $\Delta x = \frac{1}{4000}$ and $\Delta t = \frac{1}{40}$, at $\alpha = 0.2$ and $\beta = .7$.

$$f(x,t) = 2\left(t + \frac{\pi^2 t^{\beta+1}}{\Gamma(2+\beta)} + \frac{t^{\alpha+1}}{\Gamma(2+\alpha)}\right)\sin(\pi x), \tag{23}$$

with the boundary conditions $u(0,t) = u(1,t) = 0$, and the initial condition $u(x,0) = 0$.

The exact solution of Eq. (19) is $u(x,t) = t^2\sin(\pi x)$.

The behavior of the exact solution of the proposed fractional Cable equation (19) by means of the FWA-FDM is illustrated in Fig. 4.

Example 2. Consider the following initial-boundary problem of the fractional Cable equation:

$$u_t(x,t) = D_t^{1-\beta}u_{xx}(x,t) - 0.5D_t^{1-\alpha}u(x,t),$$
$$0 < x < 10, \quad 0 < t \leqslant T,$$

with $u(0,t) = u(10,t) = 0$ and $u(x,0) = 10\delta(x-5)$,

where $\delta(x)$ is the Dirac delta function.

The numerical solutions of this example are presented in Figs. 8–12.

In Fig. 8 the approximate solution of the large scale problem is shown in 3-D as an illustrative figure of the behavior of Cable equation of this example in 3-D.

In Fig. 9 different numerical solutions at different values of T are tested and figured to illustrate behavior of Cable equations for these values.

In summary, Figs. 5–7 and 10–12 show the parallel execution times, speedup and efficiencies for solving the time fractional Cable equation, for different problem size (N = 4000, 6400, 8000) and increasing number of processes (p = 1, 2,

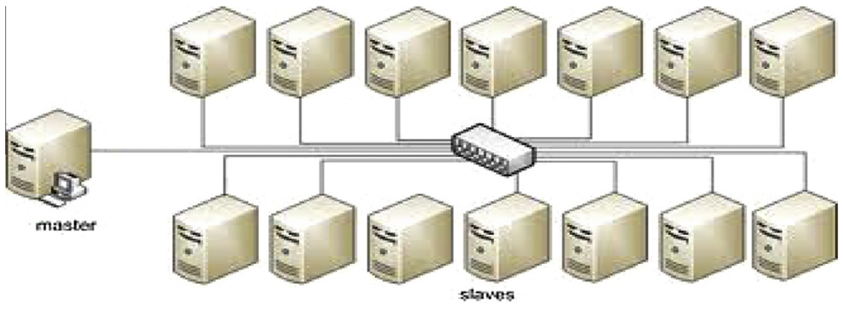

Figure 2 PC cluster description.

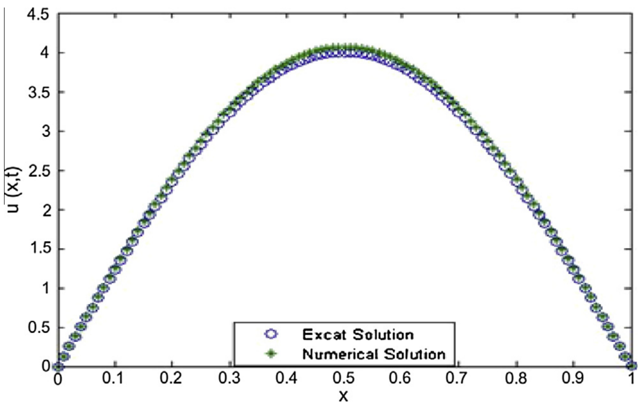

Figure 4 The behavior of the exact solution and the numerical solution of 19 at $\lambda = 0$ for $\alpha = 0.2$, $\beta = 0.7$, $\Delta x = \frac{1}{100}$, $\Delta t = \frac{1}{40}$, with $T = 2$.

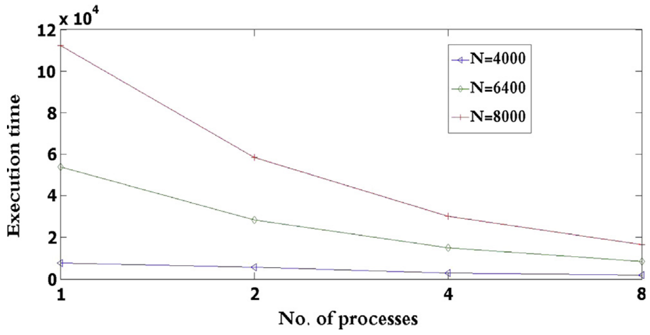

Figure 5 Scaled execution time for different problem sizes at $\alpha = 0.2$ and $\beta = 0.7$ and different number of processes.

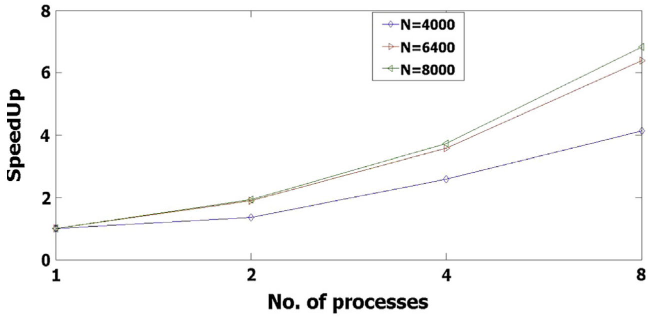

Figure 6 Scaled speedup for different problem sizes at $\alpha = .2$ and $\beta = .7$ and different number of processes.

…, 8, 16). Figs. 5 and 10 present the parallel execution time with respect to the number of processes. It can be observed that for large N (N = 6400, 8000) the parallel execution time T_p decreases with p, whereas for small problem size (N = 4000) it remains almost the same for p = 2, 4, 8 for example 1, and for p = 1, 2, 8, 16 in example 2. The difference in number of processors between the two examples is due to memory limitation of the used cluster.

Figs. 5 and 10 show experimental speedup curves for solving the problem for different sizes using increasing number of processes. As expected, for a given number of processes, the speedup increases with increasing problem size. Also, for a given problem size, the speedup does not continue to increase with the number of processes, but tends to saturate.

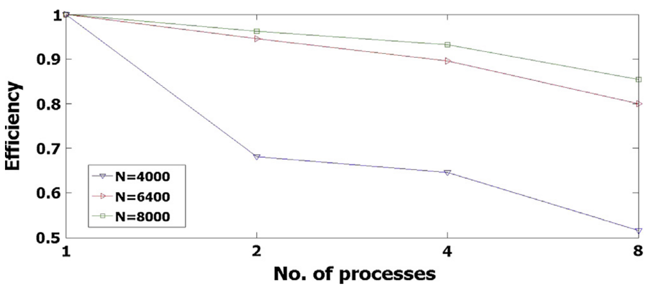

Figure 7 Scaled efficiency for different problem sizes at $\alpha = .2, \beta = .7$, and different number of processes.

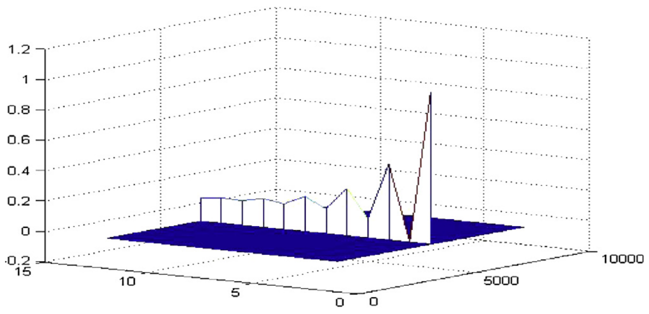

Figure 8 Approximate solution with $\Delta t = \frac{1}{40}$, $\Delta x = \frac{1}{4000}$, where $\alpha = 0.2$ and $\beta = .7$.

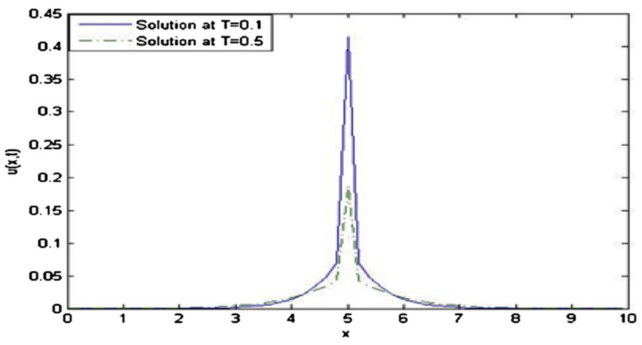

Figure 9 The exact and approximate solutions where $\lambda = 0$, $\alpha = 0.5$, $\beta = 0.5$, $\Delta x = \frac{1}{50}$, $\Delta t = \frac{1}{30}$.

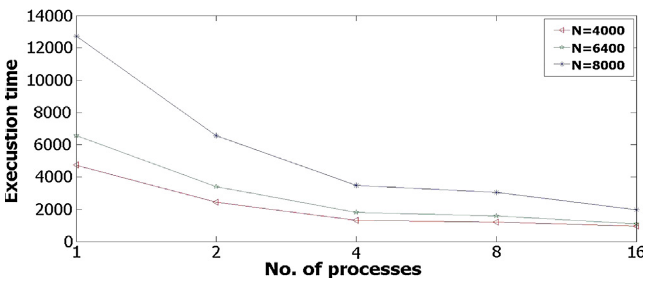

Figure 10 Scaled execution time for different problem sizes at $\alpha = .2$ and $\beta = .7$ and different number of processes.

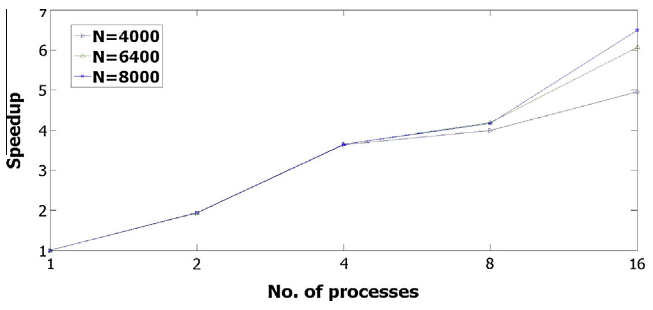

Figure 11 Scaled speedup for different problem sizes at $\alpha = .2$ and $\beta = .7$ and different number of processes.

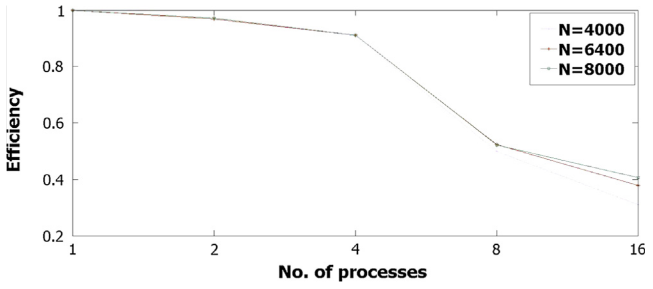

Figure 12 Scaled efficiency for different problem sizes at $\alpha = .2$ and $\beta = .7$ and different number of processes.

Figs. 6 and 11 show the efficiency curves for solving the problem for different sizes using increasing number of processes. It is also clear that efficiencies tend to decrease with the number of processes.

6. Conclusion

In this work, the fractional weighted average finite difference method FWA-FDM on cluster using Message Passing Interface (MPI) is investigated. The resultant large system of equations is studied using PCG, with the implementation of cluster computing on it. Due to the large number of neural cells and the need to the simulation of it through the Cable equation the cluster computing is an essential in this field and can help in many biological applications.

References

[1] Brown E, Wu E, Zipfel W, Webb W. Measurement of molecular diffusion in solution by multiphoton fluorescence photobleaching recovery. J Biophys 1999;77:2837–49.

[2] Feder T, Brust-Mascher I, Slattery J, Baird B, Webb W. Constrained diffusion or immobile fraction on cell surfaces: a new interpretation. J Biophys 1996;70:2767–73.

[3] Ghosh R. Mobility and clustering of individual low density lipoprotein receptor molecures on the surface of human skin fibroblasts. Ph.D. thesis. Ithaca, NY: Cornell University; 1991.

[4] Ghosh R, Webb W. Automated detection and tracking of individual and clustered cell surface low density lipoprotein receptor molecules. J Biophys 1994;66:1301–18.

[5] Simson R, Yang B, Moore S, Doherty P, Walsh F, Jacobson K. Structural mosaicism on the submicron scale in the plasma membrane. J Biophys 1998;74:297–308.

[6] Brown E, Wu E, Zipfel W, Webb W. Measurement of molecular diffusion in solution by multiphoton fluorescence photobleaching recovery. J Biophys 1999;77:2837–49.

[7] Mainardi F. Fractional diffusive waves in viscoelastic solids. Nonlinear Waves Solids 1995:93–7.

[8] Henry BI, Langlands TAM, Wearne SL. Fractional cable models for spiny neuronal dendrites. Phys Rev Lett 2008;100(12):128103.

[9] Podlubny I. Fractional differential equations. San Diego: Academic Press; 1999.

[10] Liu F, Anh V, Turner I, Zhuang P. Time fractional advection dispersion equation. Appl Math Comput 2003, p. 233–246.

[11] Liu F, Yang Q, Turner I. Stability and convergence of two new implicit numerical methods for the fractional Cable equation. J Comput Nonlinear Dyn 2011;6(1). Article ID 01109. p. 7.

[12] Quintana-Murillo J, Yuste SB. An explicit numerical method for the fractional cable equation. Int J Differen Equat 2011, p. 57–69.

[13] Langlands TAM, Henry B, Wearne S. Solution of a fractional cable equation: finite case; 2005. < http://www.maths.unsw.edu.au/applied/files/2005/amr05-33.pdf > .

[14] Langlands TAM, Henryand BI, Wearne SL. Fractional cable equation models for anomalous electrodiffusion in nerve cells: infinite domain solutions. J Math Biol 2009;59(6), p. 761–808.

[15] Rall W. Core conductor theory and Cable properties of neurons. In: Poeter R, editor. Handbook of physiology: the nervous system, vol. 1. Bethesda, Md.: American Physiological Society; 1977. p. 39–97 [chapter 3].

[16] Lubich C. Discretized fractional calculus. SIAM J Math Anal 1986;17, p. 704–719.

[17] Morton KW, Mayers DF. Numerical solution of partial differential equations. Cambridge: Cambridge University Press; 1994.

[18] Sweilam NH, Khader MM, Nagy AM. Numerical solution of two-sided space-fractional wave equation using finite difference method. J Comput Appl Math 2011;235, p. 2832–2841.

[19] Palencia E, Cuesta EA. Numerical method for an integro-differential equation with memory in Banach spaces: qualitative properties. SIAM J Numer Anal 2003;41, p. 1232–1241.

[20] Sweilam NH, Khader MM, Adel MM. On the fundamental equations for modeling neuronal dynamics. JAR J 2013.

[21] Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. J Res Nat Bur Stand 1952;49:409–36.

[22] Shewchuk JR. An introduction to the conjugate gradient method without the agonizing pain. Pittsburgh: School of Computer Science, Carnegie Mellon University; 1994.

[23] Benzi M. Preconditioning techniques for large linear systems: a survey. J Comput Phys 2002;182:418–77.

[24] O'Leary Dianne P. Notes on some methods for solving linear systems; 2007.

[25] Saad Y. Iterative methods for sparse linear systems. London: International Thomson Publ.; 1996.

[26] Thaoma R, Gudula R. Parallel programming for multicore and cluster systems. Berlin Heidelberg: Springer-Verlag; 2010.

[27] Sweilam NH, Moharram HM, Sameh Ahmed. On the parallel iterative finite difference algorithm for 2-D Poisson's equation with MPI cluster. In: The 8th international conference on INFOrmatics and systems (INFOS2012), IEEE Explorer; 2012.