

[Egyptian Informatics Journal (2015) 16, 37–44](http://dx.doi.org/10.1016/j.eij.2014.12.001)

Cairo University

Egyptian Informatics Journal

[www.elsevier.com/locate/eij](http://www.elsevier.com/locate/eij) [www.sciencedirect.com](http://www.sciencedirect.com/science/journal/11108665)

ORIGINAL ARTICLE

Cluster computing for the large scale discrete fractional Cable equation



N.H. Sweilam [\*](#_bookmark1), 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

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.

© 2015 Production and hosting by Elsevier B.V. on behalf of Faculty of Computers and Information,

KEYWORDS

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

Cairo University. This is an open access article under the CC BY-NC-ND license ([ht](#_bookmark0)[tp://creativecommons.](http://creativecommons/)

[org/licenses/by-nc-nd/4.0/).](http://creativecommons.org/licenses/by-nc-nd/4.0/)

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

\* Corresponding author.

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

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

**Production and hosting by Elsevier**

system. The number of nerve cells in the brain varies dramati- cally 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,

<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. This is an open access article under the CC BY-NC-ND license [(http://creativecommons.org/licenses/by-nc-nd/4.0/).](http://creativecommons.org/licenses/by-nc-nd/4.0/)

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 character- ized by a single parameter of the diffusion constant [[1]](#_bookmark25). 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*) ~ *t*∞. As examples, single particle tracking experiments have revealed subdiffusion (0 < *a* < 1) of proteins and lipids in a variety of cell membranes [[1–4]](#_bookmark25). Anomalous subdiffusion has also been observed in neural cell adhesion molecures [[5]](#_bookmark20). Indeed, anomalous diffusion occurs in many other physical situations, such as, transport of fluid in porous media [[6]](#_bookmark20), and the propagation of mechanical diffusive waves in vis- coelastic media [[7]](#_bookmark20).

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

continuous and *y*'(*x*) is integrable in the interval [0, *x*], then for every order 0 < *a* < 1 both the Riemann–Liouville and the Gru¨nwald–Letnikov derivatives exist and coincide for any value inside the interval This fact of fractional calculus ensures the con- sistency of both definitions for most physical applications, where the functions are expected to be sufficiently smooth [[9,10]](#_bookmark20).

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 dis- cretization 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.

1. Approximate formula for fractional derivative

Let us consider the initial-boundary value problem of the frac- tional Cable equation which is usually written in the following way (see [[11–15,20]](#_bookmark20) and the reference cited therein):

*ut*(*x*; *t*)= *D*1—*buxx*(*x*; *t*)— *lD*1—*au*(*x*; *t*); *a* < *x* < *b*; 0 < *t* ≤ *T*

*t t*

Cable equation. Very recently, a modified Cable equation was introduced for modeling the anomalous diffusion in spiny

(3)

neuronal dendrites [[8]](#_bookmark20). 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 Gru¨nwald–Letnikov fractional derivatives are given as follows [[9]](#_bookmark20):

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

1 *dn* Z *x*  *y*(*s*)

*Day*(*x*)=

C(*n* — *a*) *dxn*

0

(*x* — *sa*—*n*+1)

*ds*; *x* > 0 (1)

*x*

where 0 < *a*; *b* ≤ 1, *l* is a constant and *D*1—*c* is the fractional

derivative defined by the Riemann–Liouville operator of order 1 — *c*, where *c* = *a*; *b*. Under the zero boundary conditions

*t*

*u*(*a*; *t*)= *u*(*b*; *t*)= 0; (4)

and the following initial condition:

*u*(*x*; 0)= *g*(*x*). (5)

* 1. *Finite difference scheme for the fractional Cable equation*

In this section, we will use the FWA-FDM to obtain the dis- cretization finite difference formula of the Cable Eq. [(3)](#_bookmark2). We use the following notations: D*t* and D*x*; as the time-step length and the space-step length, respectively. The coordinates of the

mesh points are *xj* = *a* + *j*D*x* and *t*

*m*

= *m*D*t*, and the values of

where *n* is the smallest integer exceeding *a* and C(.) is the

the approximate solution *u*(*x*; *t*) on these grid points are

*u*(*xj*; *tm*)Ξ *um* ≈ *Um*. For more details about discretization in

*j j*

Gamma function. If *a* = *m* ∈ *N*, then (1) coincides with the

classical *m*th derivative *y*(*m*)(*x*).

Definition 2. The Gru¨nwald–Letnikov definition for the

fractional calculus see [[16,17]](#_bookmark21).

In the first step, the ordinary differential operators are discretized as follows [[16]](#_bookmark21):

fractional derivatives of order *a* > 0 of the function *y*(*x*) is

@*u*

*m*+1

*um*+1 — *um*

defined by:

*j*

*x*

[ ]

1 X*h*

@*t*

*xj* ;*tm* +D*t*

2

= *dtuj* 2 + *O*(D*t*)Ξ

*j* + *O*(D*t*); (6)

D*t*

*Day*(*x*)= lim

*a*

*h*→0 *h*

*k*=0

*k*

*w*(*a*)*y*(*x* — *hk*); *x* ≥ 0; (2)

and

@ *u*

2

*um* — 2*um* + *um*

*x*

*x* (*a*)

= *d um* + *O*(D*x*)2 Ξ

*j*—1

*j j*+1 + *O*(D*x*)2. (7)

where

Gru¨nwald weights which are defined by *w*(*a*) = (—1)*k*

*h* means the integer part of *h* and *wk* is the normalized

*a*

.

*k*

*k*

@*x*2

*xj* ;*tm*

*xx j*

(D*x*)2

The Gru¨nwald–Letnikov definition is simply a generalization

In the second step, the Riemann–Liouville operator is dis-

*xj* ;*tm t j*

cretized as follows:

of the ordinary discretization formula for integer order deriva- tives. The Riemann–Liouville and the Gru¨nwald–Letnikov approaches coincide under relatively weak conditions; if *y*(*x*) is

*D*1—*cu*(*x*; *t*)|

where

*t*

= *d*1—*cum* + *O* ((D*t*)*p*); (8)

*tm*

[]

*d*1—*cum* Ξ

*w*(1—*c*)*u*(*x* , *t*

(D*t*)1—*c*

1 XD*t*

*t*

*j*

*k*

*j*

*m*

*m*

*R* = *Um* + *N kw*(1—*b*) + (1 — *k*)*w* )

Xh (1—*b* i

*j*

*b*

*r*

*r*+1

*r*=0

1 X*m*

*k*=0

=

*k*=0

*wk*

)*um*—*k*, (9)

*j*—1

*j*

*j*+1

*r*

*j*

(D*t*)1—*c*

(1—*c*

— *k*D*t*)

*m*

*j*

h*U m*—*r* — 2*Um*—*r* + *Um*—*r* i — *lNa* X*w*(1—*a*)*Um*—*r* . (18)

*r*=0

where *tm* means the integer part of *tm* and for simplicity we

D*t*

D*t*

Equation [(17)](#_bookmark8) can be transferred into a matrix form as

follows:

choose *h* = D*t*. There are many choices of the weights *wk*

(*a*)

[[16,18]](#_bookmark21), so the above formula is not unique. Let us denote

(*a*)

(*m*)

1

2 *a*

(*m*)

6 *d*

(*m*)

1

*d*

*a*(*m*)

0 ... ...

0 0 0 7

3

the generating function of the weights *wk*

by *w*(*z*, *a*), i.e.,

2 2

6 . . . 7

*d* 7

*A* = 6

. .

. .

. .

0

7

*w*(*z*, *a*)=

X∞

*k*=0

*wk z* . (10)

6

(*a*)

*k*

(*m*)

*N*—2

*d*

*a*

(*m*)

*N*—2

(*m*)

(*m*)

*N*—2

(*m*)

if

*w*(*z*, *a*)= (1 — *z*)*a*, (11)

then (9) gives the backward difference formula of the first

4

where *b*

*dN*—1 *aN*—1 5

= h*b*(*m*),*b*(*m*), ... , *b*(*m*) , *b*(*m*) ],

*m*

1

2

*N*—2

*N*—1

order, which is called the Gru¨nwald–Letnikov formula. The

*m*

1

2

*N*—2

*N*—1

*j*

coefficients *w*(*a*) can be evaluated by the recursive formula

*u* = h*u*(*m*),*u*(*m*), .. . , *u*(*m*) , *u*(*m*) ], *a*(*k*) = 1 + 2*/*, *j*

*w*(*a*) = 1 —

*j*

*a* + 1

*w*(*a*) , *w*(*a*) = 1. (12)

= 1, 2, .. . , *m* — 1, *m* = 1, 2, .. . , *M* — 1, *d*(*k*) = */*.

*k k k*—1 0

*k*

For *c* = 1 the operator *D*1—*c* becomes the identity operator so that, the consistency of Eqs. [(8) and (9)](#_bookmark5) requires *w*(0) = 1,

*t*

0

and *w*(0) = 0 for *k* P 1, which in turn means that *w*(*z*, 0)= 1. Now, to obtain the finite difference scheme of the Cable Eq. [(3)](#_bookmark2), we evaluate this Equation at the intermediate point of the

*k*

grid (*xj*, *tm*):

The fractional weighted average difference scheme is Eq.

[(17)](#_bookmark8). Fortunately, Eq. [(17)](#_bookmark8) is tridiagonal system that can be solved using conjugate gradient method. In the case of *k* = 1 and *k* = 1 we have the backward Euler fractional quadrature method and the Crank–Nicholson fractional quadrature meth- ods, respectively, which have been studied e.g., in [[19]](#_bookmark22), but at *k* = 0 the scheme is called fully implicit. For the stability analy- sis see [[20]](#_bookmark23).

2

*ut*(*x*, *t*)— *D*1—*buxx*(*x*, *t*) + *lD*1—*au*(*xj*, *tm*)= 0. (13)

*t*

*xj* ,*tm*

*t*

*2.2. The PCG method*

Then, we replace the first order time-derivative by the for- ward difference formula [(6)](#_bookmark3) and replace the second order space-derivative by the weighted average of the three-point centered formula [(7)](#_bookmark4) at the times *tm* and *tm*+1

*dtum* — n*kd*1—*bdxxum* +(1 — *k*)*d*1—*bdxxum*+1o+ *ld*1—*aum* = *Tm*,

The aim now is to introduce Eq. [(17)](#_bookmark8), 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

*j t j*

*t j t j*

*j*

(14)

algorithm from 1952 can be found in [[21]](#_bookmark24). This method is used to solve linear systems of the form:

with *k* being the weight factor and *Tm* is the resulting trunca- tion error. The standard difference formula is given by

*j*

*dtUm* — {*kd*1—*bdxxUm* + (1 — *k*)*d*1—*bdxxUm*+1}+ *ld*1—*aUm* = 0. (15)

*j*

*t*

*j*

*t*

*j*

*t*

*j*

Now, by substituting from the difference operators given by [(6), (7) and (9)](#_bookmark3), we get

*Ax* = *b* (19)

With (symmetric, positive definite) and b ∈ *Rn*. In this case the solution of the linear system is equivalent to the minimum function [[22]](#_bookmark26):

*n* 1 *T T*

*E* : *R*

→ *R*, *x* →

*x Ax* — *x b*

2

*j*—1

*j*+1

*Um*+1 — *Um*

*j*

*j*

— *k*

(D*t*)1—*b*

D*t*

1. X*m*

*Um*—*r* — 2*Um*—*r* + *Um*—*r*!

Meaning *x* solves *Ax* = *b* if and only if *E* has a global mini-

mum at *x*.

1

(1—*b*)

*r*

*j*

(D*x*)2

— (1 — *k*)

(D*t*)1—*b*

*m*

*w*

*j*—1 *j j*+1

*r*=0

× *w*(1—*b*)

*r*

*r*=0

*Um*+1—*r* — 2*Um*+1—*r* + *Um*+1—*r* (D*x*)2

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 ∈ *Rn* be

1 *m*

X

!

+ *l w*(1—*a*)*Um*—*r* = 0. (16)

X

an arbitrary start vector. We search the minimum of *E* on the

line

(D*t*)1—*a*

*r j*

*r*=0

*g* : *Rn* → *R*, *a* = *x* + *ap*.

Put *Nb*

= (D*t*)*b* , *N*

(D*x*)

2

*a*

= (D*t*)*a*, *u* = (1 — *k*)*Nb*, and under some

To obtain the minimum approximately we use an iterative

simplifications we can obtain the following form:

—*uUm*+1 + (1 + 2*u*)*Um*+1 — *uUm*+1 = *R*, (17)

search with different search directions.

The conjugate gradient method works very well on matrices

*j*—1

where

1. *j*+1

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

*pT*(*Cyk* — *b*)

*pTrk rTrk*

*ak*+1 = *k*

*k*

= *k* = *k* .

the algorithm. To increase efficiency of the algorithm we use

*k*

*k*

Preconditioning technique for improving the condition num-

*pTCpk*

*pTCpk pTCpk*

ber of a matrix. By using precondition [[23]](#_bookmark27) we can iteratively solve Eq. [(19)](#_bookmark7) 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 *Ax*~ = *b*, for which *A* is chosen such that its condition number is closer to one; in other words, *A*e is close

~

e e

to the identity.

* 1. *Mathematical formulation for PCG method*

for proof of equivalence see [[24]](#_bookmark28),

The residuals *rk*+1 = *b* — *Cyk*+1 can be computed iteratively using

*rk*+1 ← *rk* — *ak*+1*Cpk*(residual)

because

*rk* — *ak*+1*Cpk* = *b* — *C*(*yk* + *apk*)= *b* — *Cyk*+1 = *rk*+1.

A relationship between the initial residual *r*~0 of Eq.[(20)](#_bookmark9) and

~ ~

Let *A*e*x*~ = *b* be the transformed system of *A*e*x*~ = *b*. The two

0

systems are related through the following relationships [[24]](#_bookmark28):

the initial residual *r*0 of *Ax* = *b*, can be found by the following:

*B*1/2*b* — *Cy*

= *r*~0, multiplying both sides of *B*1/2*b* — *B*1/2

1/2

1/2 ~

1/2

*AB*1/2 *y*

= *r*~0 by *B*—1/2, form left, we obtain *b* — *A*(*B*1/2*y* )=

*A*e = *B*

*A*, *x* = *B*

*y*, *b* = *b y*

1/2

*B*—1/2*r*~0 or *b* — *Ax*0 = *B*—1/2*r*~0. Thus *r*0 = *B*—1/2*r*~0.

Generalizing we obtain *rk* = *B*—1/2*r*~*k*.

0

0

where we picked *B* to be a symmetric positive-definite matrix

*k*

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

Similarly we can obtain *p* = *B*—1/2

*p*~*k*.

*Ax* = *b*, we have to solve the following related problem:

*Cy* = *B*1/2*b*, *x* = *B*1/2*y*.

(20)

In all the computations to follow, we will substitute *B*—1/2*r*~*k* with r*k*. By replacing *r*~*k* by *B*1/2*rk* the residual equation becomes:

*B*1/2*rk*+1 = *B*1/2*rk* — ~*ak*+1 *Cp*~*k* or *B*1/2*rk*+1

The simplest preconditioner is a diagonal matrix whose

diagonal entries are identical to those of in this paper, we apply

= *B*1/2*rk*

— ~*ak*+1

*B*1/2A*B*1/2*p*~*k*

this preconditioner, known as the diagonal preconditioning or the Jacobi preconditioning [[25]](#_bookmark29). 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

multiplying both sides, from left, by *B*—1/2 we get:

*rk*+1 = *rk* — ~*ak*+1A*B*1/2*p*~*k*, or *rk*+1 = *rk* — ~*ak*+1A*p*

*k*

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

be less than prescribed tolerance.

Since *C* is positive definite and therefore *r*

*k*+1

*Crk*–0. The

* + 1. *Part (a): compute the new iterate yk*+1 , *the search parameter ak*, *and the residual rk*+1

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

search directions can be created iteratively. With the approach *pk*+1 = *rk*+1 + *bk*+1*pk* and *p*0 = *r*0.(search direction) multiplying both sides, from left, by *B*1/2 we get:

every iteration

*B*1/2 *p*~

= *B*1/2*r*~

+ *b*~

*B*1/2 *p*~

i.e. *p*

= *B*1/2*r*~

+ *b*~ *p*

*yk*+1

← *yk*

+ *akpk*.

or *p*

*k*+1

*k*+1

= *B*1/2(*B*1/2*r*

*k*+1

)+ *b*~

1. *k*+1

*p* or *p*

= *Br*

*k*+1

+ *b*~

*k*+1 *k*

*p*

If we replace *yk*

with *B*—1/2*xk*. while calculating value of

*k*+1

*k*+1

*k*+1 *k*

*k*+1

*k*+1

*k*+1 *k*

*yk*+1, then the equation becomes:

*B*—1/2*x*

← *B*—1/2*x*

+ ~*a p*~

We also find a new formula for the improvement *b*~*k*+1

*k*+1

*k k k*

—1/2

(*B*1/2 *r*

*T* 1/2 *r* )

*rT B*1/2 *B*1/2 *rk*+1

Multiplying both sides by *B*

from the left, value of *yk*+1

*b*~ =

*k*+1 ) (*B*

*k*+1

= *k*+1 ,

is further transformed into

*k*

*k*+1

(*B*1/2

*T*

*rk* ) (*B*

1/2 *rk* )

*rTB*1/2 *B*1/2 *r*

*k*

*rT Brk*+1

*x* ← *x* + ~*a B*1/2*p*~ , or

*k*

*or b*~

= *k*+1 .

*k*+1 *k k k*

*k*+1

*rTBr*

*k*

*xk*+1 ← *xk* + ~*akp*~*k*

We can find an analytic formula for *ak*. For fixed *yk* and *pk*,

* 1. *The iterative formulas of PCG are given below*

*b*~

*xk*+1 ← *xk* + ~*akp*~*k*.

*rk*+1 ← *rk* — *ak*+1 *Cpk*.

*pk*+1 = *Brk*+1 + *b*~*k*+1 *pk*.

*T*

*k*+1 = *k*+1

*r Br*

*k*+1

// Approximate solution at step *k* +1

// Residual

// search direction:

// improvement at step *k*

*rTBrk*

.

*k*

*a*

= .

*r r*

*T*

*k*

*k*

//step length:

*k*+1

*pTCpk*

*k*

*f*(*y*

+ *a p* ) = 1 (*y*

+ *a p* )*TC*(*y*

+ *a p* )— (*y*

+ *a p* )*Tb*

*k k k*

2 *k k k*

*k k k*

*k k k*

= 1 *a*2*pTCp* + *apTCy* + —*apTb* + .. .

2 *k k k k k*

The minimum of *f* with respect to *a* occurs when the deriva- tive is zero:

*k*

*k*

*k*

*pTCy*

*k*

*k*

so

+ *apTCp*

+ —*pTb* = 0

* 1. *Parallel implementation with row-wise block-striped* [*[26]*](#_bookmark30)

The parallel algorithm is the same as the serial but some excep- tions the parallel implementation have which are

But

*dxxum*+1—*r* = *uxx* +

*j*

(D*x*)2

12

*uxxxx* +

D*t*

1. *xxt*

"*u* +

(D*x*)2

12

*uxxxxt* + .. .#

* 1. Data reading from the input\_le and dividing it across pro- cessors (using MPI\_Bcast and MPI\_Scatter).
  2. After each processor computes inner product locally, sum reduction across all processors is required (using

And

(D*t*)2

+ 8 *uxxtt* + ... ,

(D*x*)2

D*t* "

(D*x*)2 #

MPI\_Allreduce).

* 1. The vector matrix product requires gathering all the local parts of the vector into a single vector then each processor

*dxxum*+1—*r* = *uxx* +

12

*j*

(D*t*)2

*uxxxx* + 2

*uxxt* +

12 *uxxxxt* + .. .

do the multiplication(using MPI\_Allgather).

+ 8 *uxxtt* + ... ,

where the partial derivatives are evaluated at the point

The following is the parallel code fragment for the PCG for

2

solving linear systems of the form *Ax* = *b*.

(*xj*, *t*

*m*—*k*

+ D*t*)*.* Inserting these expressions into Eq. [(21)](#_bookmark10) and

taking into account Eqs. [(3) and (9)](#_bookmark2), we can get:

1 (D*x*)2

*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 else

beta = rho/oldrho

// initial direction.

// improvement at step *k* i.e. *b*

*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 *a* 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

*Tm* = *O*(*hp*)— — *k* D*tD*1—*buxxt* —

*j*

*s*

*s*

2

(D*t*)2 1

*D*1—*buxxxx*

12

— *D*1—*bu* —

(1 — *k*)*w*(1—*b*)*d u*(0) + .. . ,

8 *s xxtt*

*h*1—*b*

*m*+1 *xx j*

With *s* = *tm* i.e.

1

*Tm* = *O*(*hp*)+

— *k*

*O*(D*t*)+ *O*(D*t*)2 + *O*(D*x*)2

*j*

2

+ 1 *w*(1—*b*)*d*

*u*(0)

*h*1—*b m*+1

*xx j*

where terms of order *O*[(D*t*)*a*(D*x*)*bhp*] with a + b + p > 2 have not been included.h

1. Stability analysis and truncation error

Theorem 3.1. The truncation error of Eq. [(3)](#_bookmark2) is given by

1. 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 immedi- ately 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 *TP*, 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 *SP*, is the execution time of a sequential program divided by the execu- tion time of a parallel program that computes the same result. In particular, *SP* = *TS*/*TP*, where *TS* is the sequential execu-

*m p* 1

2 2 1

(1—*b*)

(0)

tion time and *TP* is the parallel execution time on *P* processors.

*T* = *O*(*h* )+( — *k*)*O*(D*t*)+ *O*(D*t*) + *O*(D*x*) +

*j*

2

*h*1—*b wm*+1 *dxxuj* .

The third is the efficiency. Efficiency *EP*, is a normalized mea- sure of speedup: *EP* = *SP*/*P*. Ideally, speedup should scale lin-

Proof. From the definition of truncating error given by Eq. [(14)](#_bookmark6), one gets:

early with *P*, implying that efficiency should have a constant value of 1. Of course, because of various sources of perfor- mance loss, efficiency is more typically below 1, and it dimin-

*Tm* = *d um* — {*kd*1—*bd*

*um* + (1 — *k*)*d*1—*bd*

*um*+1},

ishes as we increase the number of processors. Efficiency

*j t j*

i.e.

*Tm* = *d um* — 1

*j*

*t j*

*h*1—*b*

*tt xx j*

*m*

X h

*w*(1—*b*) (1 — *k*)*d*

*r*

*xx*

*r*=0

*tt xx j*

*um*+1—*r* + *kd*

*j*

*xx*

*um*—*r* i

greater than 1 represents superlinear speedup [[26]](#_bookmark30).

The parallel PCG has been implemented on the computer cluster of Faculty of Science Cairo University [[27]](#_bookmark31), a cluster of 17 workstations, one master and eight slaves, similar to

[Fig. 1](#_bookmark11). Each slave in the cluster has a microprocessor Intel(R)

— 1 (1 — *k*)*w*(1—*b*)*d*

*j*

*u*(0). (21)

core(TM)2 i7-2600@3,40 GHz and 8 Gb DDR3 RAM. The

*h*1—*b*

*m*+1

*xx j*

master has a microprocessor Intel(R) core(TM)2 Quad CPU Q6700 2.66 GHz and 4 Gb DDR2 RAM. The slaves are

42 N.H. Sweilam et al.

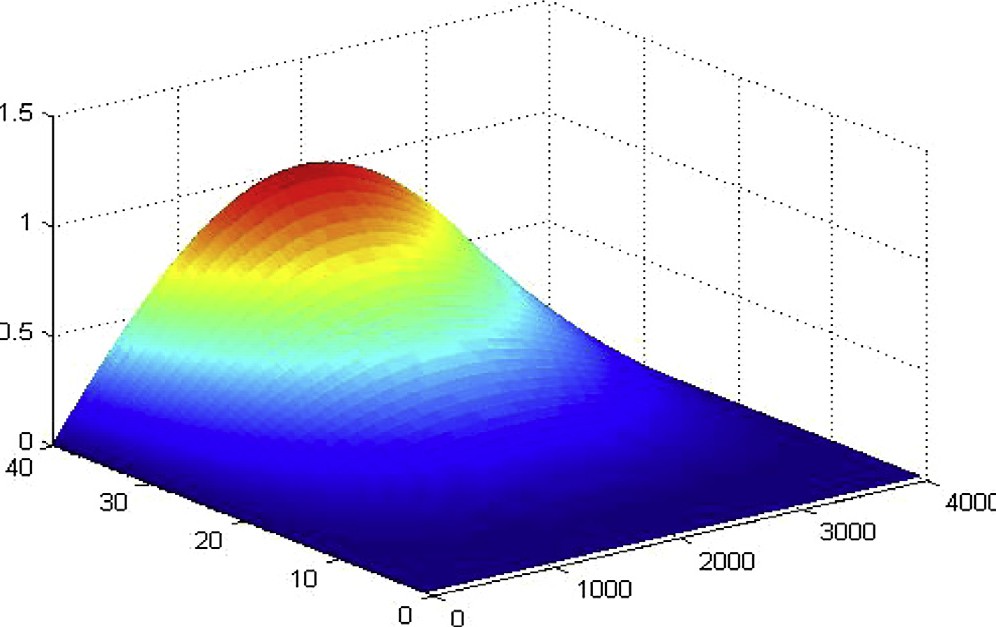


Figure 3 Approximate solution with D*x* = 1 and D*t* =  1 , at

*a* = 0.2 and *b* = .7.

4000 40

Figure 1 Data dependency of PCG algorithm.

*p*2 *tb*+1

*ta*+1

connected to the master via star local network (1 Gbps). The network consists of a switch, a network card in each worksta- tion and the corresponding Cat.6 UTP network wires (see [Fig. 2](#_bookmark13)).

*f*(*x*, *t*)= 2 *t* + +

C(2 + *b*) C(2 + *a*)

sin(*px*), (23)

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.

1. 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](#_bookmark12) the behavior of the approximate solution with D*x* = 1 , D*t* =  1 *at a* = 0.2, *b* = 0.7 is shown in a 3-D figure to display the simulation of fractional Cable equation in 3-D. While in [Fig. 3](#_bookmark12) we compare the exact solution with the numeri- cal solution.

4000 40

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

*ut*(*x*, *t*)= *D*1—*buxx*(*x*, *t*)— *D*1—*au*(*x*, *t*)+ *f*(*x*, *t*), (22)

*t t*

on a finite domain 0 < *x* < 1, with 0 6 *t* 6 *T*, 0 < *a*, *b* < 1 and the following source term:

with the boundary conditions *u*(0, *t*)= *u*(1, *t*)= 0, and the ini- tial condition *u*(*x*, 0)= 0.

The exact solution of Eq. [(19)](#_bookmark7) is *u*(*x*, *t*)= *t*2 sin(*px*).

The behavior of the exact solution of the proposed frac- tional Cable equation [(19)](#_bookmark7) by means of the FWA-FDM is illus- trated in [Fig. 4](#_bookmark14).

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

*ut*(*x*, *t*)= *D*1—*buxx*(*x*, *t*)— 0.5*D*1—*au*(*x*, *t*),

*t t*

0 < *x* < 10, 0 < *t* 6 *T*,

with *u*(0, *t*)= *u*(10, *t*)= 0 and *u*(*x*, 0)= 10*d*(*x* — 5), where *d*(*x*) is the Dirac delta function.

The numerical solutions of this example are presented in [Figs. 8–12](#_bookmark15).

In [Fig. 8](#_bookmark15) the approximate solution of the large scale prob- lem is shown in 3-D as an illustrative figure of the behavior of Cable equation of this example in 3-D.

In [Fig. 9](#_bookmark17) different numerical solutions at different values of *T* are tested and figured to illustrate behavior of Cable equa- tions for these values.

In summary, [Figs. 5–7](#_bookmark16) and [10–12](#_bookmark19) show the parallel execu- tion times, speedup and efficiencies for solving the time frac- tional Cable equation, for different problem size (*N* = 4000, 6400, 8000) and increasing number of processes (*p* = 1, 2,

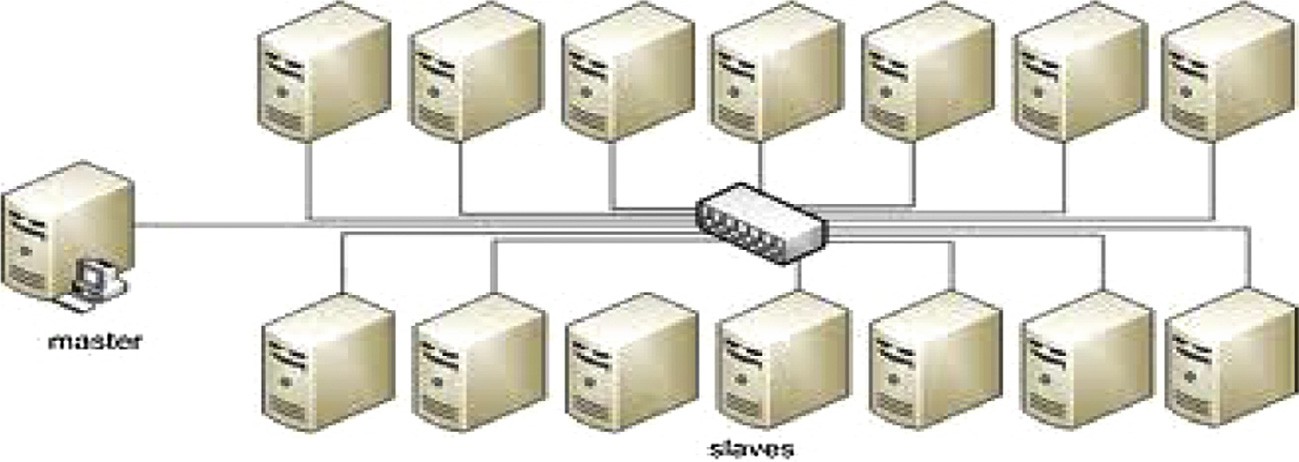
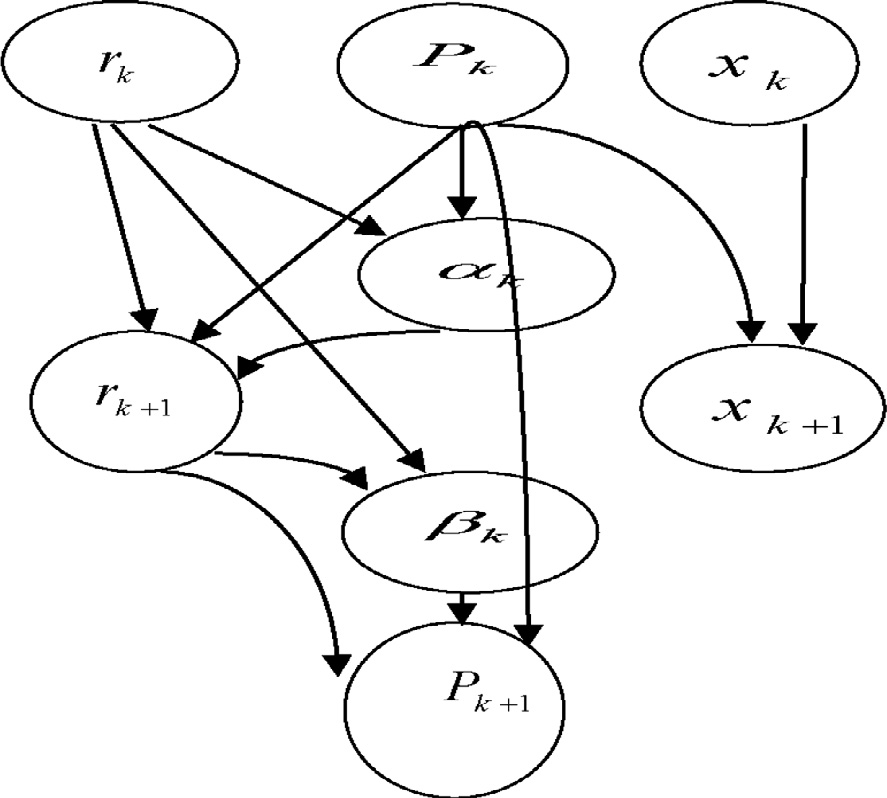


Figure 2 PC cluster description.

Cluster computing for the fractional Cable equation 43

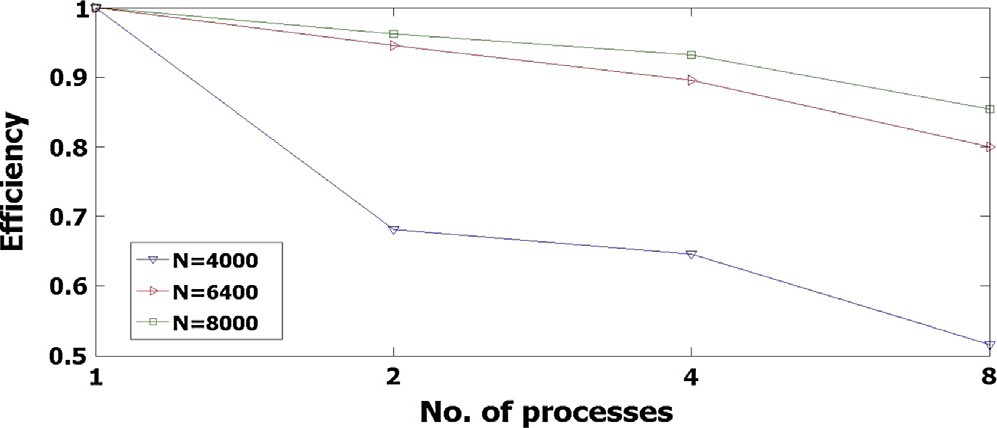
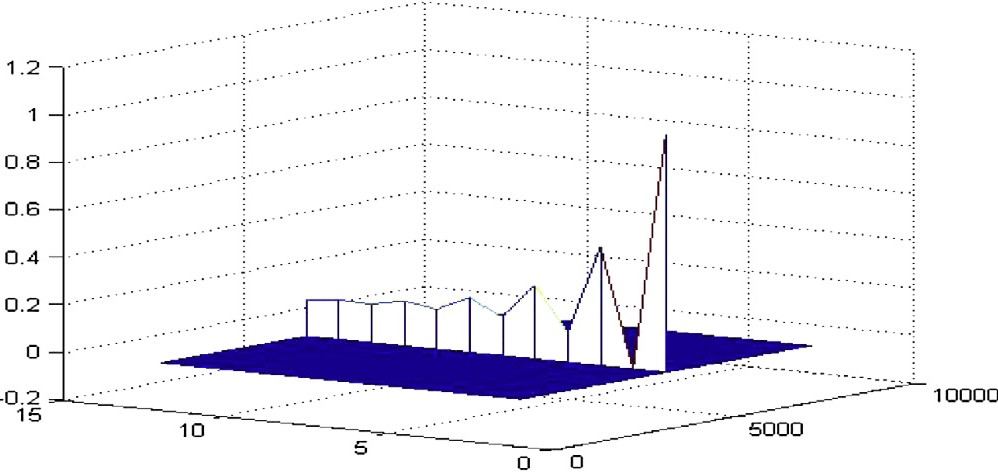


Figure 7 Scaled efficiency for different problem sizes at

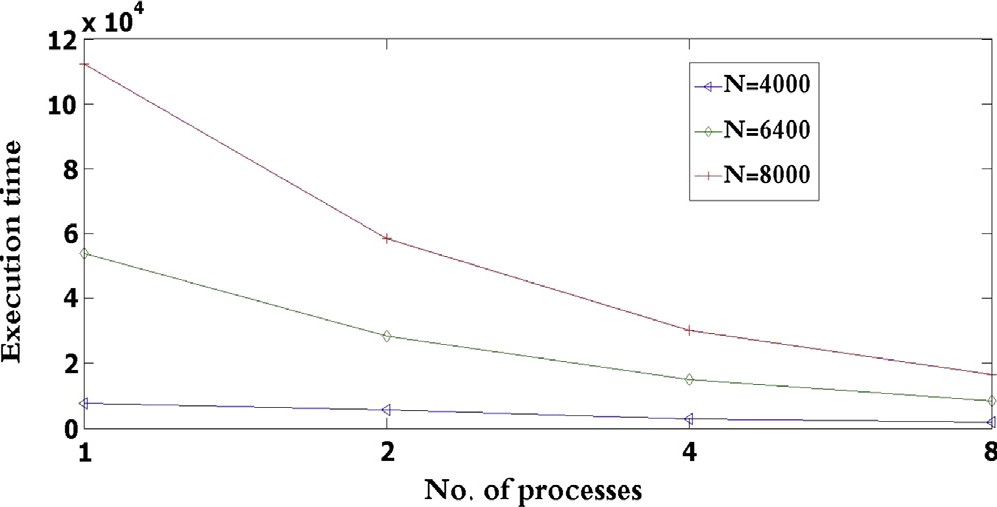
*a* = .2, *b* = .7, and different number of processes.

Figure 4 The behavior of the exact solution and the numerical solution of 19 at *k* = 0 for *a* = 0.2, *b* = 0.7, D*x* = 1 , D*t* =  1 , with

*T* = 2.

100 40

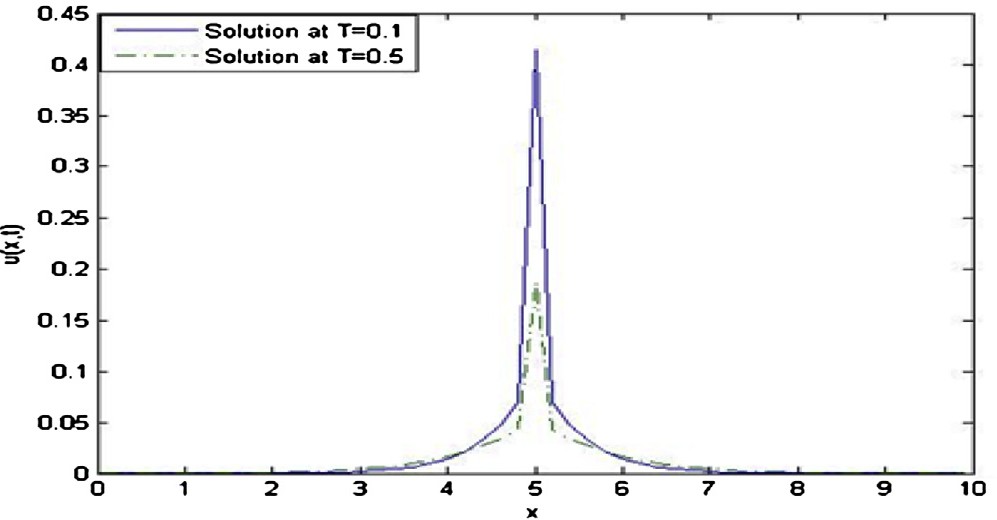
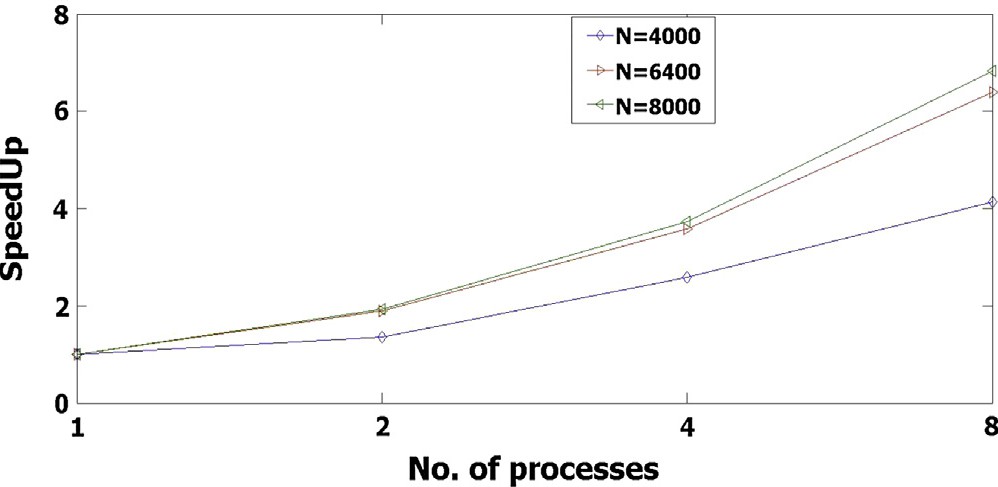
Figure 8 Approximate solution with D*t* =  1 , D*x* = 1 , where

Figure 5 Scaled execution time for different problem sizes at

*a* = 0.2 and *b* = 0.7 and different number of processes.

*a* = 0.2 and *b* = .7.

40 4000

Figure 9 The exact and approximate solutions where *k* = 0,

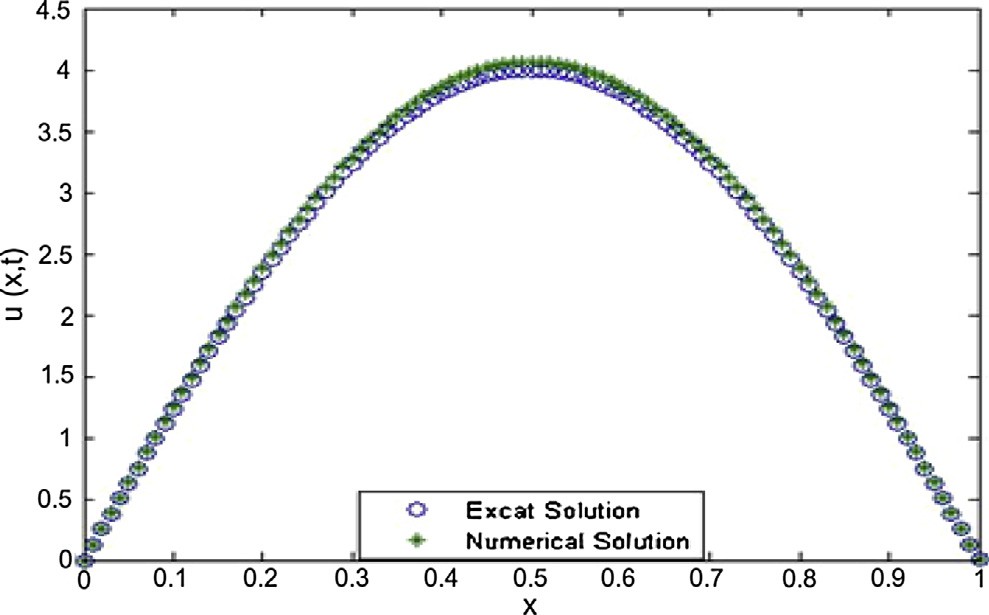
*a* = 0.5, *b* = 0.5, D*x* =  1 , D*t* =  1 .

Figure 6 Scaled speedup for different problem sizes at

*a* = .2 and *b* = .7 and different number of processes.

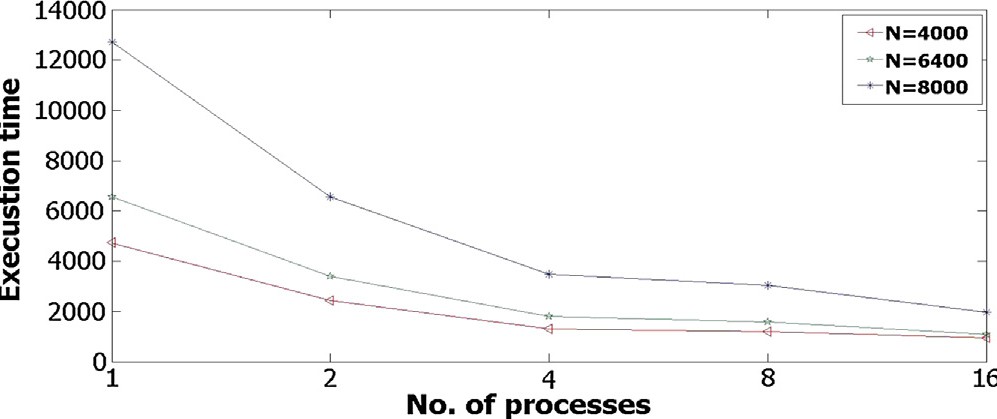
50 30

.. ., 8, 16). [Figs. 5 and 10](#_bookmark16) 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 *Tp* 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](#_bookmark16) show experimental speedup curves for solv- ing 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

Figure 10 Scaled execution time for different problem sizes at

*a* = .2 and *b* = .7 and different number of processes.

given problem size, the speedup does not continue to increase with the number of processes, but tends to saturate.

44 N.H. Sweilam et al.

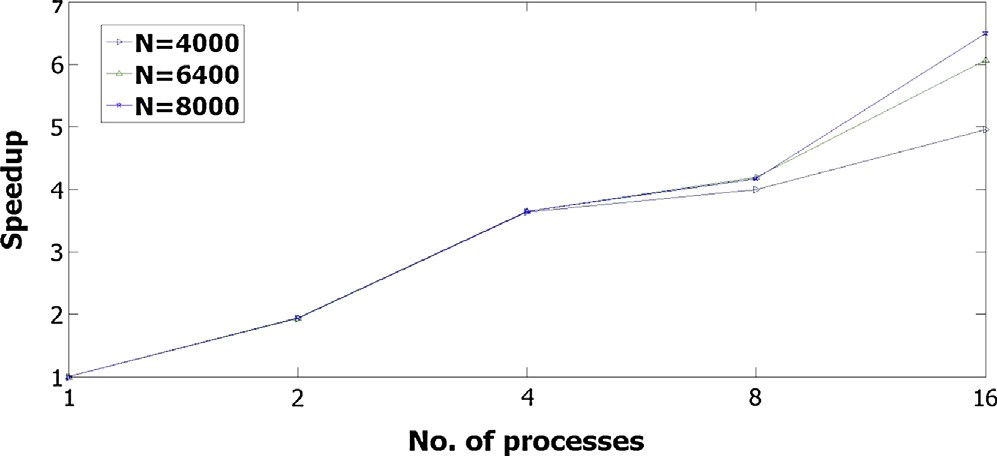


Figure 11 Scaled speedup for different problem sizes at

*a* = .2 and *b* = .7 and different number of processes.

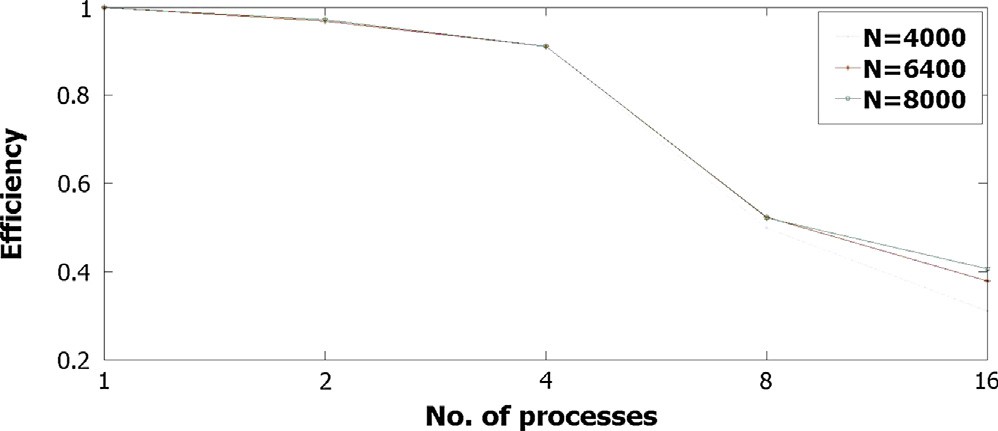


Figure 12 Scaled efficiency for different problem sizes at

*a* = .2 and *b* = .7 and different number of processes.

[Figs. 6 and 11](#_bookmark18) show the efficiency curves for solving the problem for different sizes using increasing number of pro- cesses. It is also clear that efficiencies tend to decrease with the number of processes.

1. 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](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0005) [diffusion in solution by multiphoton fluorescence photobleaching](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0005) [recovery. J Biophys 1999;77:2837–49](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0005).
2. [Feder T, Brust-Mascher I, Slattery J, Baird B, Webb W.](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0010) [Constrained diffusion or immobile fraction on cell surfaces: a](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0010) [new interpretation. J Biophys 1996;70:2767–73](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0010).
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](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0020) [individual and clustered cell surface low density lipoprotein](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0020) [receptor molecules. J Biophys 1994;66:1301–18](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0020).
5. [Simson R, Yang B, Moore S, Doherty P, Walsh F, Jacobson K.](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0025) [Structural mosaicism on the submicron scale in the plasma](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0025) [membrane. J Biophys 1998;74:297–308](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0025).
6. [Brown E, Wu E, Zipfel W, Webb W. Measurement of molecular](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0030) [diffusion in solution by multiphoton fluorescence photobleaching](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0030) [recovery. J Biophys 1999;77:2837–49](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0030).
7. [Mainardi F. Fractional diffusive waves in viscoelastic solids.](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0035) [Nonlinear Waves Solids 1995:93–7](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0035).
8. [Henry BI, Langlands TAM, Wearne SL. Fractional cable models](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0040) [for spiny neuronal dendrites. Phys Rev Lett 2008;100(12):128103](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0040).
9. [Podlubny I. Fractional differential equations. San Diego:](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0045) [Academic Press; 1999](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0045).
10. [Liu F, Anh V, Turner I, Zhuang P. Time fractional advection](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0050) [dispersion equation. Appl Math Comput 2003, p. 233–246](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0050).
11. [Liu F, Yang Q, Turner I. Stability and convergence of two new](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0055) [implicit numerical methods for the fractional Cable equation. J](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0055) [Comput Nonlinear Dyn 2011;6(1). Article ID 01109. p. 7](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0055).
12. [Quintana-Murillo J, Yuste SB. An explicit numerical method for](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0060) [the fractional cable equation. Int J Differen Equat 2011, p. 57–69](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0060).
13. Langlands TAM, Henry B, Wearne S. Solution of a fractional cable equation: finite case; 2005. <[http://www.maths.unsw.edu.](http://www.maths.unsw.edu.au/applied/files/2005/amr05-33.pdf) [au/applied/files/2005/amr05-33.pdf](http://www.maths.unsw.edu.au/applied/files/2005/amr05-33.pdf)>.
14. [Langlands TAM, Henryand BI, Wearne SL. Fractional cable](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0070) [equation models for anomalous electrodiffusion in nerve cells:](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0070) [infinite domain solutions. J Math Biol 2009;59(6), p. 761–808](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0070).
15. [Rall W. Core conductor theory and Cable properties of neurons.](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0075) [In: Poeter R, editor. Handbook of physiology: the nervous](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0075) [system, vol. 1. Bethesda, Md.: American Physiological Society;](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0075) [1977. p. 39–97 [chapter 3]](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0075).
16. [Lubich C. Discretized fractional calculus. SIAM J Math Anal](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0080) [1986;17, p. 704–719](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0080).
17. [Morton KW, Mayers DF. Numerical solution of partial dif-](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0085) [ferential equations. Cambridge: Cambridge University Press;](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0085) [1994](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0085).
18. [Sweilam NH, Khader MM, Nagy AM. Numerical solution of](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0090) [two-sided space-fractional wave equation using finite difference](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0090) [method. J Comput Appl Math 2011;235, p. 2832–2841](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0090).
19. [Palencia E, Cuesta EA. Numerical method for an integro-](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0095) [differential equation with memory in Banach spaces: qualitative](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0095) [properties. SIAM J Numer Anal 2003;41, p. 1232–1241](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0095).
20. [Sweilam NH, Khader MM, Adel MM. On the fundamental](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0100) [equations for modeling neuronal dynamics. JAR J 2013](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0100).
21. [Hestenes MR, Stiefel E. Methods of conjugate gradients for](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0105) [solving linear systems. J Res Nat Bur Stand 1952;49:409–36](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0105).
22. [Shewchuk JR. An introduction to the conjugate gradient method](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0110) [without the agonizing pain. Pittsburgh: School of Computer](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0110) [Science, Carnegie Mellon University; 1994](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0110).
23. [Benzi M. Preconditioning techniques for large linear systems: a](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0115) [survey. J Comput Phys 2002;182:418–77](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0115).
24. O’Leary Dianne P. Notes on some methods for solving linear systems; 2007.
25. [Saad Y. Iterative methods for sparse linear systems. London:](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0125) [International Thomson Publ.; 1996](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0125).
26. [Thaoma R, Gudula R. Parallel programming for multicore and](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0130) [cluster systems. Berlin Heidelberg: Springer-Verlag; 2010](http://refhub.elsevier.com/S1110-8665(14)00047-4/h0130).
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.