VERSITA

Fractional Calculus & Applied Analysis
An International Journal for Theory and Applications

VOLUME 16, NUMBER 3 (2013)
(Print) ISSN 1311-0454
(Electronic) ISSN 1314-2224

RESEARCH PAPER

A PARALLEL ALGORITHM FOR THE RIESZ FRACTIONAL REACTION-DIFFUSION EQUATION WITH EXPLICIT FINITE DIFFERENCE METHOD

Chunye Gong [1,2,3], **Weimin Bao** [1,2], **Guojian Tang** [1]

Abstract

The fractional reaction-diffusion equations play an important role in dynamical systems. Indeed, it is time consuming to numerically solve differential fractional diffusion equations. In this paper, we present a parallel algorithm for the Riesz space fractional diffusion equation. The parallel algorithm, which is implemented with MPI parallel programming model, consists of three procedures: preprocessing, parallel solver and postprocessing. The parallel solver involves the parallel matrix vector multiplication and vector vector addition. As to the authors' knowledge, this is the first parallel algorithm for the Riesz space fractional reaction-diffusion equation. The experimental results show that the parallel algorithm is as accurate as the serial algorithm. The parallel algorithm on single Intel Xeon X5540 CPU runs 3.3-3.4 times faster than the serial algorithm on single CPU core. The parallel efficiency of 64 processes is up to 79.39% compared with 8 processes on a distributed memory cluster system.

MSC 2010: Primary 26A33; Secondary 33E12, 34A08, 34K37, 35R11, 60G22

Key Words and Phrases: Riesz fractional differential equation, parallel algorithm, Reaction-Dispersion Equation, MPI, explicit finite difference method

© 2013 Diogenes Co., Sofia
pp. 654–669 , DOI: 10.2478/s13540-013-0041-8

1. Introduction

The reaction-diffusion equations, as examples of fractional order differential equations, play an important role in dynamical systems of mathematics, physics, chemistry, bioinformatics, finance and other research and applied areas. For example, fractional diffusion equations have been used to represent anomalous diffusion processes such as diffusion of water in biological tissues (see [19]) and diffusion pumps in vacuum technology.

Because most of the problems for fractional differential equations cannot be solved analytically, more and more works focus on their numerical solutions, as for example, these proposed in [12, 34]. The numerical schemes for such solutions include the finite difference method [34], finite element method [3, 11], finite volume method [16], Fourier transform method [21], spectral method [20], multigrid method [28], iterative method [9], boundary element method [17], etc. The dimension of such fractional order problems ranges usually from one to three, see [32, 26, 22].

Recently, the interest in fractional reaction-diffusion equations has substantially increased, see e.g. [5, 7, 27, 6]. Chen and Liu considered explicit finite-difference approximations [5, 7] and implicit difference approximation [6] for the Riesz space fractional reaction-dispersion equation. Meerschaert et al. [24] presented finite difference methods for two-sided space-fractional partial differential equations and discussed the stability, consistency, and convergence of these methods.

It is time consuming to numerically solve fractional differential equations for high spatial dimension or great time integration. Short memory principle [29, 33] and parallel computing [10] can be used to overcome this difficulty. Parallel computing is a form of computation in which many arithmetic logic operations are carried out concurrently. Large scale applications in science and engineering such as particle transport [13, 14, 18, 8] rely on parallel computing. Kai Diethelm [10] implemented the fractional version of the second-order Adams-Bashforth-Moulton method on a parallel computer and discussed the precise nature of the parallelization concept. Parallel computation has already appeared in some studies on fractional differential equations, but until today their power for approximating fractional derivatives and solving fractional differential equations has not been fully recognized.

This paper focuses on the Riesz space fractional reaction-diffusion equation:

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t} = -u(x,t) + {}_xD_0^{\alpha} u(x,t) + q(x,t) \\ u(x,0) = g(x), \ x \in (x_L, x_R) \\ u(x_L,t) = u(x_R,t) = 0 \end{cases} \tag{1.1}$$

with $1 < \alpha \leq 2$ and $0 \leq t \leq T$. If α equals 2, Eq. (1.1) is the classical diffusion equation. Both $u(x,t)$and $g(x)$ are real-valued and sufficiently well-behaved functions, and ${}_xD_0^\alpha u(x,t)$ is the Riesz space fractional derivative.

We present a parallel algorithm for Eq. (1.1) and implement the parallel algorithm with MPI programming model. The parallel algorithm keeps the same accuracy and convergence as the serial algorithm and good scalability is validated on a distributed memory cluster system.

The rest of this paper is organized as follows. The background of the numerical solution and parallel computing is introduced in Section **2**. In Section **3**, we present the parallel solution in detail. The experimental results and discussion are collected in section **4**. Section **5** gives conclusions and future work.

2. Background

2.1. Numerical solution

The Riesz space fractional derivative of order α, ${}_xD_0^\alpha u(x,t)$ is defined by analytic continuation in the whole range $1 < \alpha \leq 2$ (see e.g. [6, 15])

$$ {}_xD_0^\alpha u(x,t) = -I^{-\alpha} = -c(I_+^{-\alpha} + I_-^{-\alpha}) \tag{2.1}$$

with

$$ c = \frac{1}{2cos(\alpha\pi/2)},\ I_\pm^{-\alpha} = \frac{(d^2x)}{dx^2} I_\pm^{2-\alpha}, \tag{2.2}$$

where the Weyl integrals $I_\pm^{2-\alpha}$ are defined as (see [30])

$$ \begin{cases} (I_+^\alpha \psi)(x) = \dfrac{1}{\Gamma(\alpha)} \displaystyle\int_0^x (x-\xi)^{\alpha-1} \psi(\xi) d\xi, \\ (I_-^\alpha \psi)(x) = \dfrac{1}{\Gamma(\alpha)} \displaystyle\int_x^L (\xi - x)^{\alpha-1} \psi(\xi) d\xi, \end{cases} \tag{2.3}$$

$\psi(x) \in L_1(-\infty, +\infty)$.

The shifted Grünwald finite difference formula [25] for spatial α-order fractional derivative is

$$ \begin{cases} (I_+^\alpha \psi)(x)|_{x=x_i} = \lim\limits_{i\to\infty} \dfrac{1}{h^\alpha} \displaystyle\sum_{k=0}^{i} g_k \psi(x-(k-1)h), \\ (I_-^\alpha \psi)(x)|_{x=x_i} = \lim\limits_{M\to\infty} \dfrac{1}{h^\alpha} \displaystyle\sum_{k=0}^{M-i} g_k \psi(x+(k-1)h), \end{cases} \tag{2.4}$$

where $h = (x_R - x_L)/M$ and $0 \le i \le M$. The following approximation from Meerschaert and Tadjeran [23] can be used:

$$\begin{cases} (I_+^\alpha \psi)(x)|_{x=x_i} = \dfrac{1}{h^\alpha} \displaystyle\sum_{k=0}^{i} g_k \psi(x-(k-1)h) + O(h), \\ (I_-^\alpha \psi)(x)|_{x=x_i} = \dfrac{1}{h^\alpha} \displaystyle\sum_{k=0}^{M-i} g_k \psi(x+(k-1)h) + O(h), \end{cases} \tag{2.5}$$

where $g_0 = 1, g_k = (-1)^k \frac{\alpha(\alpha-1)\ldots(\alpha-k+1)}{k!}, k = 1, 2, 3, \ldots$ is the normalized Grünwald weight.

Define $t_n = n\Delta t, x_i = ih$ for $0 \le n \le N, 0 \le i \le M$, where M and N are positive integer, $\Delta t = \frac{T}{N}, h = \frac{x_R - x_L}{M}$ are time step and space step, respectively. Assume u_i^n is the numerical approximation to $u(x_i, t_n)$ and f_i^n is the numerical approximation to $q(x_i, t_n)$. Adopting an Euler method in time at level $t = t_n$ (and x $= x_i$) and substituting the above the expressions into Eq. (1.1), the following explicit difference approximation can be obtained:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -u_i^n + \frac{c}{h^\alpha}[\sum_{k=0}^{i+1} g_k u_{i+1-k}^n + \sum_{k=0}^{M-i+1} g_k u_{i-1+k}^n] + f_i^n. \tag{2.6}$$

Eq. (2.6) results in a linear system of equations

$$U^{n+1} = AU^n + Q^n, \tag{2.7}$$

where $U^n = (u_1^n, u_2^n, \ldots, u_{M-1}^n)^T$, $Q^n = (\Delta t f_1^n, \Delta t f_2^n, \ldots, \Delta t f_{M-1}^n)^T$ with $q_i^n = \Delta t f_i^n$ $(1 \le i < M)$, and $A = (a_{ij})_{(M-1)\times(M-1)}$ is a matrix of coefficient, defined by

$$a_{i,j} = \begin{cases} -\dfrac{\Delta t}{ch^\alpha} g_{i+1-j} \text{ for } 1 \le j < i-1 \\ -\dfrac{\Delta t}{ch^\alpha}(g_0 + g_2) \text{ for } j = i-1 \\ (1-\Delta t) - 2\dfrac{\Delta t}{ch^\alpha} g_1 \text{ for } j = i \\ -\dfrac{\Delta t}{ch^\alpha}(g_0 + g_2) \text{ for } j = i+1 \\ -\dfrac{\Delta t}{ch^\alpha} g_{j-i+1} \text{ for } i+1 < j < M-1 \end{cases}. \tag{2.8}$$

2.2. Parallel computing

The parallel computing is a form of computation in which many arithmetic logic operations are carried out concurrently. It is used to solve problems faster than by serial computing or problems which single computing

node does not have enough capability to deal with. So the parallel computing has two characteristics: high computing speed and huge computing capability. The parallel computing involves parallel algorithms/applications, programming models, system softwares and hardware. Driven by the capabilities and limitations of modern semiconductor manufacturing, the computing industry is currently undergoing a massive shift towards parallel computing, see e.g. [2, 4]. The types of parallelism are bit-level, instruction-level, data-level and task level parallelism.

The scalability is defined as the parallel solution time varies with the process configuration for a fixed computational load per process. The computational load is determined by the problem size, which is defined by M and N ($h = \frac{x_R - x_L}{M}$ and $\Delta t = \frac{T}{N}$). The parallel efficiency [1], P_{e2}, of a problem size S_2 on N_2 processes compared with a problem size S_1 on N_1 processes can be defined by

$$P_{e2} = \frac{S_2}{N_2 T_2} / \frac{S_1}{N_1 T_1}, \tag{2.9}$$

where T_1 and T_2 are the runtimes of N_1 and N_2 processes.

The programming model is a bridge between the hardware and applications. Message Passing Interface (MPI) [31] is a standard and portable library of communications subroutines for parallel programming designed to function on a wide variety of parallel computers. MPI is a de facto standard for communication of parallel computing in both the area of academic and industry.

3. Parallel algorithm

The parallel algorithm consists of three parts. The first part is preprocessing, which prepares the initial variables, matrices, vectors and so on. The second part is the parallel solver. The parallel solver focuses on the iteration of time step, which is the most time consuming parts of the total numerical solution. The third part is postprocessing, which outputs the final results and so on.

3.1. Preprocessing and postprocessing

The preprocessing includes initialization of parallel environment, distribution of computing task, allocation of local memory space and initialization of variables and arrays. The matrices $A_{(M-1)\times(M-1)}$ and $Q_{(M-1)\times N}$ are prepared before the computation of Eq. (2.7). For example, matrix A can be found according to Eq. (2.8).

The postprocessing is simple. The results for the exact solution are performed. The max absolute error of the exact and parallel solutions is

computed and outputted. Both the results of the exact and parallel solution are saved in files which are necessary for the plot. Other operations of postprocessing includes free memory space and stop the parallel environment.

3.2. Design of the parallel solver

In order to get U^{n+1}, the right-sided computation of Eq. (2.7) should be performed. There are one matrix vector multiplication and one vector-vector addition in the right-sided computation:

(1) The matrix vector multiplication is $V^1 = AU^n$.
(2) The vector vector addition is $V^1 + V^2$ and a new vector $U^{n+1} = V^1 + V^2$ is got. Here V^2 stands for Q^n.

In fact, the matrix vector multiplication and vector-vector addition can be combined with each other like Eq. (2.7). The vectors V^1, V^2 are not real arrays and only used to describe the computing procedure.

We assume that L processes are used to solve Eq. (2.7). In order to perform these computation simultaneously, these vector and matrix should be divided up into several parts. If there are L processes, the matrices A, U, Q are divided into L parts following the rows of them. So the vectors U^n, U^{n+1} are separated into L parts.

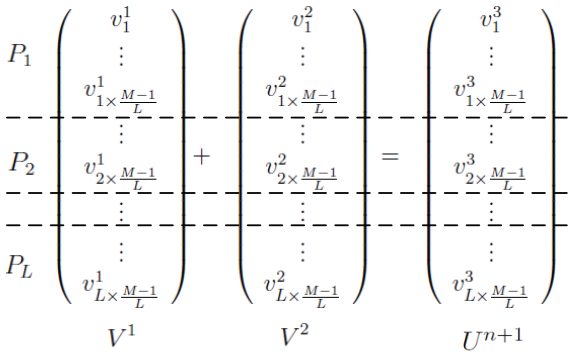

FIGURE 1. Parallel vector vector addition.

Assuming M–1 is divisible by L, each vector is divided into L subvectors and each subvector has $\kappa = \frac{M-1}{L}$ elements. Each matrix is divided into L submatrices and each submatrix is a $\frac{M-1}{L} \times (M-1)$ matrix. The parallel vector-vector addition is described in Fig. 1. The dashed lines in Fig. 1 stand for the split lines of vectors. P_1, P_2, P_L stand for the first, second and L^{th} process. From Fig. 1, we can see each process just deal with the locate subvectors.

The parallel matrix vector multiplication of AU^n for process P_i is shown in Eq. (3.1) as follows:

$$A^i_{\kappa\times(M-1)} \times U^n = \begin{pmatrix} a^i_{1,1} & a^i_{1,2} & \cdots & a^i_{1,M-1} \\ a^i_{2,1} & a^i_{2,2} & \cdots & a^i_{2,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ a^i_{\kappa,1} & a^i_{\kappa,2} & \cdots & a^i_{\kappa,M-1} \end{pmatrix} \begin{pmatrix} u^n_1 \\ u^n_2 \\ \vdots \\ u^n_{M-1} \end{pmatrix}. \tag{3.1}$$

The matrix vector multiplication for process is P_i straightforward, but the vector U^n is distributed among all process. So all parts of the vector U^n should be gathered to process P_i as described in Fig. 2. The 'Left' stands for the distribution of the vector U^n and the 'Right' stands for the gathered vector U^n in process P_i. Because i ranges from 1 to L, the gather operation of U^n should be performed for every process. This is an all-gather operation for all processes to get the whole U^n.

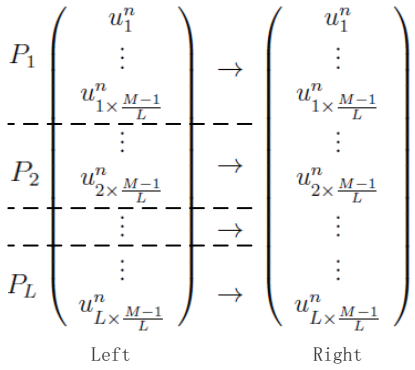

FIGURE 2. Gather operation of U^n.

The storage of matrices A is column first in *Fortran*. In order to get good memory access performance, the matrix A needs to be transposed into A^T. This transposition does not need to perform explicitly. The elements of the matrix A are computed and directly stored into the matrix A^T. So the matrix A is not stored in the memory.

3.3. Implementation

The parallel solution uses the mechanisms of process level parallelism. The process level parallelism is a kind of task level parallelism. The parallel algorithm for Eq. (1.1) is described in Algorithm 1.

The preprocessing involves line 1 to 5. The parallel solver involves line 6 to 11. The postprocessing involves line 12 to 13 and other additional operations are not shown in Algorithm 1. Because the memory storage strategy

of *Fortran* is column first, the matrix A needs to be transposed. The matrix transpose will enhance the data locality and improve the application performance. The all gather operation uses the function MPI_Allgather_ of MPI. So each time step (line 8-11) needs one communication.

Algorithm 1: Parallel solution for Riesz space fractional reaction-diffusion equation

```
1  Init parallel environment
2  Allocate local memory A^T, U, Q ... .
3  Init variables and arrays
4  for all MPI processes(1 to L) do in parallel
5      compute A^T directly
6      record time T_1
7      compute U^1 with g(x) in Eq. 1.1
8      for n = 1 to N − 1 do
9          all gather U^n
10         compute local U^{n+1} ← A^T × U^n + Q^n
11     record time T_2
12 output T_2 − T_1 and U^N ...
13 stop parallel environment
```

The locate computation of Eq. (2.7), or line 11 of Algorithm 1, is described in Algorithm 2. $(M-1)/L$ is the size of vector V^2. The matrix vector multiplication and vector-vector addition are similar for Algorithm 2.

Algorithm 2: Computation of Eq. 2.8

```
   input  : Δt, Q^n, M, L, U^n, U^{n+1}, A^T
   output: U^{n+1}
1  define locate variable ξ
2  for i = 1 to (M−1)/L do
3      ξ=0
4      for j = 1 to M − 1 do
5          ξ ← ξ + A^T_{j,i} × U^n_j
6      U^{n+1}_i = ξ
```

CPU	Intel Xeon E5540, 4 cores, 2.53 GHz
Operating System	Kylin server version 3.1
Compiler	mpif90, Intel Fortran version 11.1
Communication	MPICH2 version 1.3rc2

TABLE 1. Technical specifications of the experiment platform

4. Experimental results and discussion

The experiment platform is a cluster with DSM (distributed memory system) architecture. One computing node consists of two Intel Xeon E5540 CPUs. The specifications of the cluster are listed in Table 1. The code runs on double precision floating point operations and is compiled by the mpif90 compiler with level three optimization (-O3).

4.1. Numerical example

The following space fractional ($\alpha = 1.8$) differential equation [5] is considered

$$\frac{\partial u(x,t)}{\partial t} = -u(x,t) +_x D_0^{\alpha} u(x,t) + q(x,t)(0 \le t \le 1) \tag{4.1}$$

where $x \in (0,2)$, with the source

$$\begin{aligned} q(x,t) = \frac{e^{-t}}{2\cos(\alpha\pi/2)} \times [\frac{24(x^{2.3} + (2-x)^{2.3})}{\Gamma(3.3)} \\ - \frac{24(x^{1.3} + (2-x)^{1.3})}{\Gamma(2.3)} + \frac{8(x^{0.3} + (2-x)^{0.3})}{\Gamma(1.3)}], \end{aligned} \tag{4.2}$$

the initial condition

$$u(x,0) = x^2(2-x)^2 \quad \text{for } 0 \le x \le 2 \tag{4.3}$$

and the boundary conditions

$$u(0,t) = 0, \ u(2,t) = 0 \text{ for } 0 < t \le 1. \tag{4.4}$$

The exact solution of Eq. (4.1) is

$$u(x,t) = e^{-t}x^2(2-x)^2.$$

4.2. Accuracy of the parallel solution

The parallel solution compares well with the exact analytic solution to the fractional partial differential equation in this test case of Eq. (4.1), shown in Fig. 3. The Δt and h are $\frac{0.1}{256}$ and $\frac{2.0}{129}$. The maximum absolute error is 5.59×10^{-4}.

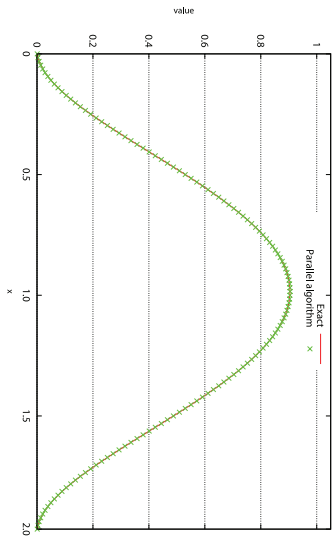

FIGURE 3. Comparison of the exact solution to the solution of the parallel algorithm at time t = 0.1.

In fact, the accuracy and convergence of the parallel solution are the same as the serial solution. The final results are independent of using how many processes when meeting the condition the number of processes is less than $M-1$.

4.3. Performance

For fixed $N = 64$, the performance comparison between single process and four processes (single CPU) is shown in Fig. 4. The dimension size of matrix A in Eq. (2.7) is $M-1$, which is the x-coordinate of Fig. 4. $M-1$ ranges from 24576 to 57344. With $M-1 = 24576$, the runtime of one process, four processes and eight processes are 58.73, 17.58 and 12.42 seconds, respectively. Compared with one process, four and eight processes perform the same computation 3.34 and 4.73 times faster, respectively. With $M-1 = 57344$, the runtime of one process is 342.80 seconds and the runtime of four and eight processes are 104.27 and 65.58 seconds. The speedup of four and eight processes are up to 3.29 and 5.23, compared with one process. The performance of four processes is about 3.30 times higher than the performance of single process with fixed $N = 64$. The performance of eight processes is 4.55-5.23 times higher than the performance of single process with fixed $N = 64$.

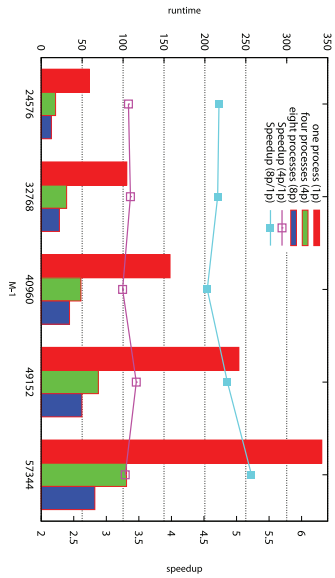

FIGURE 4. Performance comparison among single process, four processes and eight processes on E5540 with fixed N.

For fixed $M-1 = 40960$, the performance comparison between single process and four processes is shown in Fig. 5. For single process, the row number of matrix A is 40960. For four processes, the row number of local matrix A is 10240. N ranges from 32 to 512. With $N = 32$, the runtime of one process is 80.05 seconds and the runtime of four processes is 23.65 seconds. The speedup is 3.38. With $N = 512$, the runtime of one process is 1490.17 seconds and the runtime of four processes is 415.81 seconds. The speedup is 3.58. So the performance of four processes is about 3.40 times higher than the performance of single process with fixed M.

The scalability of the parallel Algorithm 1 on the large scale cluster system is shown in Fig. 6. The technical specifications of the cluster system is listed in Table 1. N is fixed with 32 for all conditions. $M-1$ varies from 55296, 78208, 95784, 110592 to 156416 for 8, 16, 24, 32 and 64 processes. Each process has the similar computational load at all conditions. The runtime of 8 processes is 31.00 seconds and the runtime of 64 processes is 39.05 seconds. The parallel efficiency of 64 processes is 79.39% compared with 8 processes.

4.4. Discussion

The numerical methods for the stand seconder-order partial differential equations yield sparse and banded coefficient matrices. The numerical methods for the space fractional differential equations yield fairly dense or

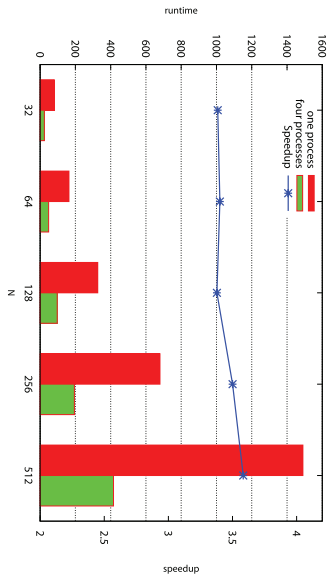

FIGURE 5. Performance comparison between single process and four processes on E5540 with fixed M.

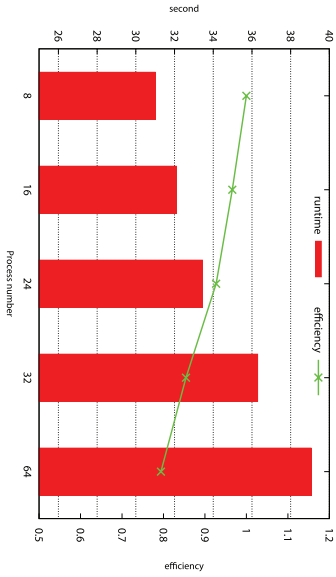

FIGURE 6. Scalability of the parallel algorithm on a cluster system.

even completely full matrices. For example, the coefficient matrix of the space fractional advection-dispersion equations (not Riesz) [24] is a lower

triangular matrix with one superdiagonal and Eq. 2.8 is a completely full matrix. For time fractional equations, the non-locality of the time derivative has effects on the computational efficiency and the SpMV (sparse matrix vector multiplication) plays an important role.

The goal of this article is not to develop a new numerical method with better approximation properties but rather to accelerate the existing explicit finite difference method with MPI programming model. While the parallel algorithm is described only for a very special example equation (Riesz fractional) and a very special discretization scheme (explicit difference method), the parallel scheme could be applied to a much more general class of one-dimensional Riesz space fractional equation such as Riesz space fractional diffusion equation, Riesz space fractional advection-diffusion equation. This paper focuses on using a distributed memory architecture with the MPI programming model for communication, the approach could be used for other parallelization paradigms too, such as shared memory parallelization with OpenMP (Open Multi-Processing) or the use of GPU (Graphics Processing Unit).

5. Conclusions

In this article, we present a parallel algorithm for one-dimensional Riesz space fractional differential equation with explicit difference method. The initialization of matrices is pre-performed before the iteration of time step. The iteration of time step consists of one matrix vector multiplication and one vector vector addition. The design detail of these matrix, vector operations are presented. The parallel algorithm is implemented with MPI programming model. The experimental results show that the parallel implementation is as accurate as serial implementation and can scales well on large scale distributed memory cluster system. So the power of parallel computing for solving fractional differential equations should be recognized.

Acknowledgements

This research work is supported by the National Natural Science Foundation of China under Grant No.60970033, also by 973 Program of China under Grant No.61312701001. We would like to thank the anonymous reviewers for their helpful comments, the in-depth discussion with Prof. Kai Diethelm, and also the help of our collaborators J. Liu, X. Sun, L. Wang.

References

[1] G. K. Ananth Grama, Anshul Gupta, V. Kumar, *Introduction to Parallel Computing*. 2nd Ed., Addison-Wesley (2003).

[2] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, et al., A view of the parallel computing landscape. *Communications of the ACM* **52** (2009), 56–67.

[3] K. Burrage, N. Hale, D. Kay, An efficient implicit FEM scheme for fractional-in-space reaction-diffusion equations. *SIAM J. on Scientific Computing* **34** (2012), 2145–2172.

[4] B. Catanzaro, N. Sundaram, K. Keutzer, Fast support vector machine training and classification on graphics processors. In: *Proc. 25th Internat. Conf. on Machine Learning ACM* (2008), 104–111.

[5] J. Chen, F. Liu, Analysis of stability and convergence of numerical approximation for the Riesz fractional reaction-dispersion equation (in Chinese). *J. of Xiamen University (Natural Science)* **45** (2006), 466–469.

[6] J. Chen, F. Liu, Stability and convergence of an implicit difference approximation for the space Riesz fractional reaction-dispersion equation. *Numerical Mathematics, A Journal of Chinese Universities (EN Ser.)* **16** (2007), 253.

[7] J. Chen, F. Liu, I. Turner, V. Anh, The fundamental and numerical solutions of the Riesz space fractional reaction-dispersion equation. *ANZIAM J.* **50** (2008), 45–57.

[8] G. Colomer, R. Borrell, F. Trias, I. Rodrguez, Parallel algorithms for s_n transport sweeps. *J. of Computational Physics* **232** (2012), 118–135.

[9] C. Dhaigude, V. Nikam, Solution of fractional partial differential equations using iterative method. *Fract. Calc. Appl. Anal.* **15**, No 4 (2012), 684–699; DOI: 10.2478/s13540-012-0046-8; at http://link.springer.com/journal/13540.

[10] K. Diethelm, An efficient parallel algorithm for the numerical solution of fractional differential equations. *Fract. Calc. Appl. Anal.* **14**, No 3 (2011), 475–490; DOI: 10.2478/s13540-011-0029-1; at http://link.springer.com/journal/13540.

[11] N. Ford, J. Xiao, Y. Yan, A finite element method for time fractional partial differential equations. *Fract. Calc. Appl. Anal.* **14**, No 3 (2011), 454–474; DOI: 10.2478/s13540-011-0028-2; at http://link.springer.com/journal/13540.

[12] G. hua Gao, Z. zhong Sun, Y. nan Zhang, A finite difference scheme for fractional sub-diffusion equations on an unbounded domain using artificial boundary conditions. *J. of Computational Physics* **231** (2012), 2865–2879.

[13] C. Gong, J. Liu, L. Chi, H. Huang, J. Fang, Z. Gong, GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method. *J. of Computational Physics* **230** (2011), 6010–6022.
[14] C. Gong, J. Liu, H. Huang, Z. Gong, Particle transport with unstructured grid on GPU. *Computer Physics Communications* **183** (2012), 588–593.
[15] R. Gorenflo, F. Mainardi, Approximation of levy-feller diffusion by random walk. *J. for Analysis and its Applications* **18** (1999), 231–246.
[16] H. Hejazi, T. Moroney, F. Liu, A finite volume method for solving the two-sided time-space fractional advection-dispersion equation. In: *Proc. FDA'12 – 5th Symposium on Fractional Differentiation and Its Applications*, Hohai University, 2012.
[17] J. T. Katsikadelis, The BEM for numerical solution of partial fractional differential equations. *Comput. Math. Appl.* **62** (2011), 891–901.
[18] D. J. Kerbyson, M. Lang, S. Pakin, Adapting wave-front algorithms to efficiently utilize systems with deep communication hierarchies. *Parallel Computing* **37** (2011), 550–561.
[19] M. Köpf, C. Corinth, O. Haferkamp, T. Nonnenmacher, Anomalous diffusion of water in biological tissues. *Biophysical Journal* **70** (1996), 2950–2958.
[20] C. Li, F. Zeng, F. Liu, Spectral approximations to the fractional integral and derivative. *Fract. Calc. Appl. Anal.* **15**, No 3 (2012), 383–406; DOI: 10.2478/s13540-012-0028-x; at http://link.springer.com/journal/13540.
[21] J. Lima, R. C. de Souza, The fractional Fourier transform over finite fields. *Signal Processing* **92** (2012), 465–476.
[22] S. Lu, F. Molz, G. Fix, Possible problems of scale dependency in applications of the three-dimensional fractional advection-dispersion equation to natural porous media. *Water Resour. Res.* **38** (2002), 1165.
[23] M. Meerschaert, C. Tadjeran, Finite difference approximations for fractional advection–dispersion flow equations. *J. of Computational and Applied Mathematics* **172** (2004), 65–77.
[24] M. Meerschaert, C. Tadjeran, Finite difference approximations for two-sided space-fractional partial differential equations. *Applied Numerical Mathematics* **56** (2006), 80–90.
[25] K. Miller, B. Ross, *An Introduction to the Fractional Calculus and Fractional Differential Equations*. John Wiley & Sons, New York (1993).
[26] Y. nan Zhang, Z. zhong Sun, Alternating direction implicit schemes for the two-dimensional fractional sub-diffusion equation. *J. of Computational Physics* **230** (2011), 8713–8728.

[27] N. Ozdemir, D. Avci, B. Iskender, The numerical solutions of a two-dimensional space-time Riesz-Caputo fractional diffusion equation. *Intern. J. of Optimization and Control: Theories & Applications (IJOCTA)* **1**, (2011), 17–26.

[28] H.-K. Pang, H.-W. Sun, Multigrid method for fractional diffusion equations. *J. of Computational Physics* **231** (2012), 693–703.

[29] I. Podlubny, *Fractional Differential Equations*, Academic Press, San Diego, CA (1999).

[30] S. Samko, A. Kilbas, O. Maričev, *Fractional Integrals and Derivatives*, Gordon and Breach Science Publ., Yverdon (1993).

[31] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, S. Huss-Lederman, *MPI: The Complete Reference*, MIT Press, Cambridge, MA – USA (1995).

[32] C. Tadjeran, M. M. Meerschaert, H.-P. Scheffler, A second-order accurate numerical approximation for the fractional diffusion equation. *J. of Computational Physics* **213** (2006), 205–213.

[33] Y. Xu, Z. He, The short memory principle for solving abel differential equation of fractional order. *Computers & Mathematics with Applications* **62** (2011), 4796–4805.

[34] S. B. Yuste, J. Quintana-Murillo, A finite difference method with non-uniform timesteps for fractional diffusion equations. *Computer Physics Communications* **183** (2012), 2594–2600.

[1] *College of Aerospace Science and Engineering*
National University of Defense Technology
Changsha 410073, CHINA
e-mail: gongchunye@nudt.edu.cn, gongchunye@gmail.com

Received: October 13, 2012
Revised: March 19, 2013

[2] *Science and Technology on Space Physics Libratory*
Beijing 10076, CHINA

[3] *School of Computer Science*
National University of Defense Technology
Changsha 410073, CHINA

Please cite to this paper as published in:

Fract. Calc. Appl. Anal., Vol. **16**, No 3 (2013), pp. 654–669;
DOI:10.2478/s13540-013-0041-8