Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 219580, 8 pages
http://dx.doi.org/10.1155/2014/219580

Research Article

A Parallel Algorithm for the Two-Dimensional Time Fractional Diffusion Equation with Implicit Difference Method

Chunye Gong,[1,2,3] Weimin Bao,[1,2] Guojian Tang,[1] Yuewen Jiang,[4] and Jie Liu[3]

[1] *College of Aerospace Science and Engineering, National University of Defense Technology, Changsha 410073, China*
[2] *Science and Technology on Space Physics Laboratory, Beijing 100076, China*
[3] *School of Computer Science, National University of Defense Technology, Changsha 410073, China*
[4] *Department of Engineering Science, University of Oxford, Oxford OX2 0ES, UK*

Correspondence should be addressed to Chunye Gong; gongchunye@gmail.com

Received 9 January 2014; Accepted 6 February 2014; Published 12 March 2014

Academic Editors: F. Liu, A. Sikorskii, and S. B. Yuste

It is very time consuming to solve fractional differential equations. The computational complexity of two-dimensional fractional differential equation (2D-TFDE) with iterative implicit finite difference method is $O(M_x M_y N^2)$. In this paper, we present a parallel algorithm for 2D-TFDE and give an in-depth discussion about this algorithm. A task distribution model and data layout with virtual boundary are designed for this parallel algorithm. The experimental results show that the parallel algorithm compares well with the exact solution. The parallel algorithm on single Intel Xeon X5540 CPU runs 3.16–4.17 times faster than the serial algorithm on single CPU core. The parallel efficiency of 81 processes is up to 88.24% compared with 9 processes on a distributed memory cluster system. We do think that the parallel computing technology will become a very basic method for the computational intensive fractional applications in the near future.

1. Introduction

Building fractional mathematical models for specific phenomenon and developing numerical or analytical solutions for these fractional mathematical models are very hot in recent years. Fractional diffusion equations have been used to represent different kinds of dynamical systems [1]. But the fractional applications are rare. One reason for rare fractional applications is that the computational cost of approximating for fractional equations is too much heavy. The idea of fractional derivatives dates back to the 17th century. A fractional differential equation is a kind of equation which uses fractional derivatives. Fractional equations provide a powerful instrument for the description of memory and hereditary properties of different substances.

There has been a wide variety of numerical methods proposed for fractional equations [2, 3], for example, finite difference method [4–7], finite element method [8, 9], spectral method [10, 11], and meshless techniques [12]. Zhuang and Liu [4] presented an implicit difference approximation for two-dimensional time fractional diffusion equation (2D-TFDE) on a finite domain and discussed the stability and convergence of the method. The numerical result of an example agrees well with their theoretical analysis. Tadjeran and Meerschaert presented a numerical method, which combines the alternating directions implicit (ADI) approach with a Crank-Nicolson discretization and a Richardson extrapolation to obtain an unconditionally stable second-order accurate finite difference method, to approximate a two-dimensional fractional diffusion equation [13]. Two ADI schemes based on the L_1 approximation and backward Euler method are considered for the two-dimensional fractional subdiffusion equation [14].

It is very time consuming to numerically solve fractional differential equations for high spatial dimension or big time integration. Short memory principle [15] and parallel computing [16, 17] can be used to overcome this difficulty. Parallel computing is used to solve computation intensive applications simultaneously [18–21]. Large scale applications in science and engineering such as particle transport [22–24],

different linear and nonlinear systems [25], nonnumerical intelligent algorithm [26], and computational fluid dynamics [27] can rely on parallel computing. Diethelm [17] 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. This is the first attempt for parallel computing on fractional equations. Following that, Gong et al. [16] presented a parallel algorithm for one-dimensional Riesz space fractional diffusion equation with explicit finite difference method. The numerical solution of Riesz space fractional equations has global dependence on grid points, which means the approximation of a grid point will depend on the approximation of all grid points in one time step. The numerical solution of time fractional equations has global dependence on time steps, which means that the approximation of a grid point will depend on the approximation of the grid point in all time steps. Global dependence means the nonlocal property of fractional deviates on time or space. Explicit method is easy to be parallelized but is restrict by its stability condition. Implicit method is hard to be solved by Gauss elimination method and often uses the iterative scheme. Until today, the power of parallel computing for high dimensional and time fractional differential equations has not been tried.

This paper focuses on the two-dimensional time fractional diffusion equation studied by Zhuang and Liu [4]:

$$\begin{aligned}\frac{\partial^\alpha u(x,y,t)}{\partial t^\alpha} &= a(x,y,t)\frac{\partial^2 u(x,y,t)}{\partial x^2} \\ &\quad + b(x,y,t)\frac{\partial^2 u(x,y,t)}{\partial y^2} + f(x,y,t), \\ u(x,y,0) &= \phi(x,y), \quad (x,y)\in\Omega, \\ u(x,y,t)|_{\partial\Omega} &= 0, \quad t\in[0,T],\end{aligned} \tag{1}$$

where $\Omega = \{(x,y) \mid 0 \le x \le L_x, 0 \le y \le L_y,\ a(x,y,t) > 0, b(x,y,t) > 0\}$. The fractional derivative is in the Caputo form.

2. Background: Numerical Solution

The fractional derivative of $f(t)$ in the Caputo sense is defined as [15]

$$\frac{\partial^\alpha f(t)}{\partial t^\alpha} = \frac{1}{\Gamma(1-\alpha)}\int_0^t \frac{f'(\xi)}{(t-\xi)^\alpha}d\xi \quad (0<\alpha<1). \tag{2}$$

If $f'(t)$ is continuous bounded derivatives in $[0,T]$ for every $T > 0$, we can get

$$\begin{aligned}\frac{\partial^\alpha f(t)}{\partial t^\alpha} &= \lim_{\xi\to 0,\, n\xi=t} \xi^\alpha \sum_{i=0}^{n}(-1)^i \binom{\alpha}{i} \\ &= \frac{f(0)\,t^{-\alpha}}{\Gamma(1-\alpha)} + \frac{1}{\Gamma(1-\alpha)}\int_0^t \frac{f'(\xi)}{(t-\xi)^\alpha}d\xi.\end{aligned} \tag{3}$$

Define $\tau = T/N$, $h_x = L_x/M_x$, $h_y = L_y/M_y$, $t_n = n\tau$, $x_i = ih_x$, and $y_j = jh_y$, for $0 \le n \le N$, $0 \le i \le M_x$, and $0 \le j \le M_y$. Let $u_{i,j}^n$, φ_i^n, $f_{i,j}^n$, $\phi_{i,j}$, $a_{i,j}^n$, and $b_{i,j}^n$ be the numerical approximation to $u(x_i, y_j, t_n)$, $f(x_i, y_j, t_n)$, $\phi(x_i, y_j)$, $a(x_i, y_j, t_n)$, and $b(x_i, y_j, t_n)$. We can get the implicit approximating scheme [4] for (1):

$$\begin{aligned}&u_{i,j}^{n+1} - u_{i,j}^n + \sum_{s=1}^{n} b_s\left(u_{i,j}^{n+1-s} - u_{i,j}^{n-s}\right) \\ &\quad = \mu_1\Gamma(2-\alpha)\,a_{i,j}^{n+1}\left(u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}\right) \\ &\qquad + \mu_2\Gamma(2-\alpha)\,b_{i,j}^{n+1}\left(u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}\right) \\ &\qquad + \tau^\alpha\Gamma(2-\alpha)\,f_{i,j}^{n+1},\end{aligned} \tag{4}$$

where $b_s = (s+1)^{1-\alpha} - s^{1-\alpha}$ $(s = 0, 1, 2, \ldots, N)$, $\mu_1 = \tau^\alpha/h_x^2$, and $\mu_2 = \tau^\alpha/h_y^2$. The h_x and h_y are the step size along X and Y directions defined above.

3. Parallel Algorithm

3.1. Analysis. Let $c_1 = c_1(i,j,k) = \mu_1\Gamma(2-\alpha)a_{i,j}^{n+1}$, and let $c_2 = c_2(i,j,k) = \mu_2\Gamma(2-\alpha)b_{i,j}^{n+1}$; (4) can be rewritten as

$$\begin{aligned}&-c_1\left(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}\right) + \left(1 + 2c_1 + 2c_2\right)u_{i,j}^{n+1} \\ &\quad - c_2\left(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\right) \\ &\quad = u_{i,j}^n - \sum_{s=1}^{n} b_s u_{i,j}^{n+1-s} + \sum_{s=1}^{n} b_s u_{i,j}^{n-s} + \tau^\alpha\Gamma(2-\alpha)\,f_{i,j}^{n+1}.\end{aligned} \tag{5}$$

The explicit schemes are conditionally stable and need very small τ for high dimensional problems for both classical and fractional equations. The implicit schemes are unconditionally stable but need to get the inverse of the coefficient matrix. Sometimes the sparse coefficient matrix is too large, making a direct method too difficult to use. So, the iterative method can be used to avoid matrix inverse:

$$\begin{aligned}&u_{i,j}^{n+1,k+1} \\ &\quad = \frac{1}{1 + 2c_1 + 2c_2} \\ &\qquad \times \left(c_1\left(u_{i+1,j}^{n+1,k} + u_{i-1,j}^{n+1,k}\right) + c_2\left(u_{i,j+1}^{n+1,k} + u_{i,j-1}^{n+1,k}\right) + u_{i,j}^n\right. \\ &\qquad\quad \left. - \sum_{s=1}^{n} b_s u_{i,j}^{n+1-s} + \sum_{s=1}^{n} b_s u_{i,j}^{n-s} + \tau^\alpha\Gamma(2-\alpha)\,f_{i,j}^{n+1}\right)\end{aligned} \tag{6}$$

until $\Delta u = |u_{i,j}^{n+1,k+1} - u_{i,j}^{n+1,k}|$ is smaller than a predefined threshold ϵ. $u_{0\to M_x, 0\to M_y}^{n+1,k+1}$ are the iterative variables. $u_{0\to M_x, 0\to M_y}^{n}$ are the known variables for the unknown $n+1$ time step.

It is very time consuming to solve the 2D-TFDE by iterative method of (6). For determining N, M_x, M_y and assuming if there are K iterations for each time step on average, there are about $M_xM_y(N^2/2 + 1.5N + 6KN)$ arithmetical

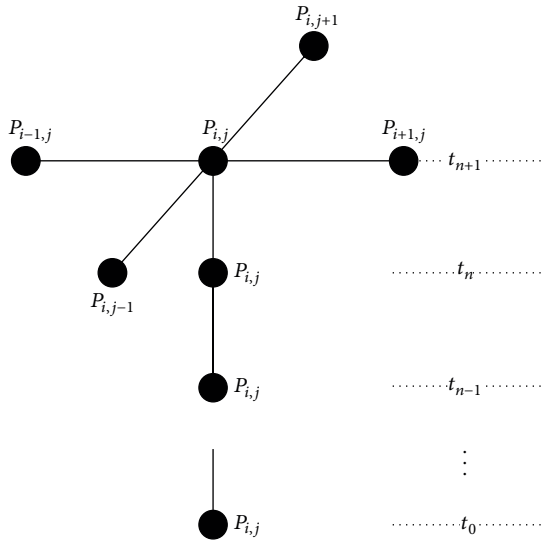

Figure 1: The data dependence of 2D-TFDE of grid point $P_{i,j}$ of time step t_{n+1}.

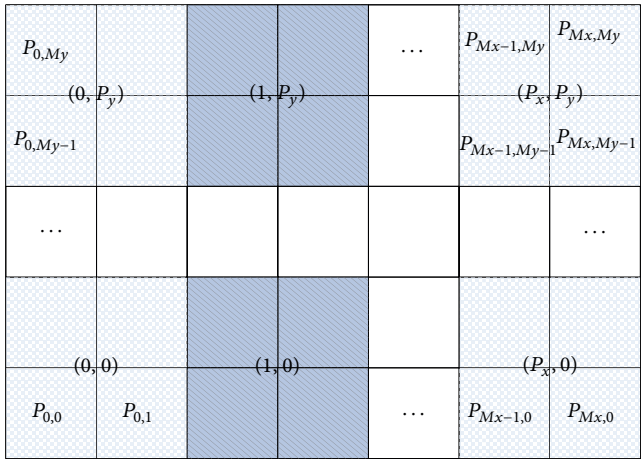

Figure 2: The two-dimensional task distribution model for 2D-TFDE.

logical operations ignoring the computation of the coefficients. So, the computational complexity is $O(M_x M_y N^2)$, which is much more heavy than the classical integer order 2D partial differential equations $O(M_x M_y N)$.

Besides the heavy computational cost, the memory space requirement is the other problem. Because each unknown time step needs to use all the values of the previous time steps, all the values of $u_{0\to M_x,0\to M_y}^{0\to N}$ need to be stored into the memory space. When N is big enough, the memory complexity is $O(M_x M_y N)$, which is far bigger than the classical integer order 2D partial differential equations $O(M_x M_y)$.

The computation of (6) can be divided into two parts.

(i) $\text{Part1}_{i,j} = u_{i,j}^n - \sum_{s=1}^n b_s u_{i,j}^{n+1-s} + \sum_{s=1}^n b_s u_{i,j}^{n-s} + \tau^\alpha \Gamma(2-\alpha) f_{i,j}^{n+1}$. The unknown value $u_{i,j}^{n+1,k+1}$ of grid point $P_{i,j}$ at the time step $n+1$ relies on the value of grid point $P_{i,j}$ at all previous time steps of $\text{Part1}_{i,j}$.

(ii) $\text{Part2}_{i,j} = c_1(u_{i+1,j}^{n+1,k} + u_{i-1,j}^{n+1,k}) + c_2(u_{i,j+1}^{n+1,k} + u_{i,j-1}^{n+1,k})$. The unknown value $u_{i,j}^{n+1,k+1}$ of grid point $P_{i,j}$ relies on the value of $P_{i+1,j}, P_{i-1,j}, P_{i,j+1}, P_{i,j-1}$.

The data dependence of 2D-TFDE is shown in Figure 1. $u_{i,j}^{n+1}$ relies on the neighboring grid points at the same time step and the same position of all the previous time steps.

3.2. Task Distribution Model and Data Layout. The task distribution of the total computation should be designed on distributed memory systems, with the goal of making the total computations as efficient as possible. There are three main issues in choosing a task distribution model for these computations:

(i) load balance: ensure splitting of the computations reasonably evenly among all computing processors/processes throughout the time stepping;

(ii) less communication: the task distribution model should keep the communication among different computing processes as less as possible;

(iii) convenient programming: the parallel algorithm based on the task distribution model should not change the serial algorithm too much.

The goal of keeping attention on these issues is achieving high execution efficiency and high scalability of the parallel algorithm on distributed memory systems for 2D-TFDE.

Refer to (6). $\text{Part2}_{i,j}$ computation has no data dependence. $\text{Part1}_{i,j}$ computation has data dependence among neighboring grid points. There are mainly two kinds of task distribution models. The first one is one-dimensional distribution (ODD): splitting the domain of all grid points along the X or Y direction on average. The task distribution model of the parallel algorithm [16] for the one-dimensional Riesz space fractional equation is ODD. The parallel algorithm based on ODD will not change the serial algorithm much and the load balance is guaranteed. If task is divided along X direction and M_y is very big, the communication will influence the scalability of the parallel algorithm. The second one is two-dimensional distribution (TDD): splitting the domain of all grid points along the X and Y direction on average. So, the computing processes have a two-dimensional grid layout, with process id (p_i, p_j) and $0 \le p_i \le P_x$, $0 \le p_j \le P_y$. P_x, P_y are the dimension size of the processes grid. The task distribution with TDD is shown in Figure 2.

With the TDD, the data layout is described in Figure 3. Each subdomain with a process may have less than four virtual boundaries to receive the boundary data from its nearest neighbors. The virtual boundary is shown with dotted lines. The process $(p_x, P_y - 1)$ $(0 \le p_x \le P_x)$ has four virtual boundaries. The process (p_x, P_y) only has three virtual boundaries since there is no process that stays on its right hand. A virtual boundary may have several layer grid points, which depends on the discrete scheme on space. In this paper, there is only one layer grid point for a virtual boundary with (4). In every iteration of (6), the processes exchange

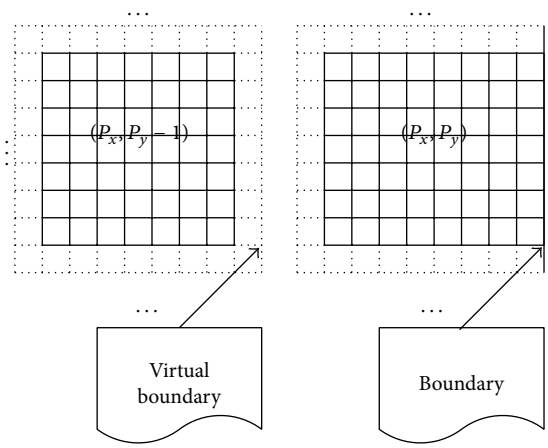

Figure 3: Data layout for 2D-TFDE.

the data near the virtual boundaries shown in Figure 3. After the exchange, every process performs its own computation according to (6).

3.3. Implementation. The parallel algorithm for 2D-TFDE uses the mechanisms of process level parallelism. The process level parallelism is a kind of task level parallelism. The parallel algorithm for (1) is described in Algorithm 1.

Each process only allocates its local memory. Assuming M_x, M_y are divisible by P_x, P_y, the process with four virtual boundaries will allocate $(M_x/P_x + 2)(M_y/P_y + 2)N$ memory space for array u. The calculation of process id has three steps:

step 1: get the MPI global id ID;

step 2: $p_y = \lfloor \mathrm{ID}/P_x \rfloor$;

step 3: $p_x = \mathrm{ID} - P_x p_y$.

The computations of $c_1(i, j)$, $c_2(i, j)$, $f_{i,j}$, and so forth depend on the particular functions of coefficient and source terms. Performing these computations, every time step is a good choice. If these computations are performed out of the main loop (lines 9–32), a lot of memory space is required. If these computations are performed in the "While" loop (lines 16–32), it is too time consuming. The u^0 stands for the zero time step $u^0_{i,j}$ and v stands for $v_{i,j}$. $I_{i,j}$ means the iteration $1 \le i \le M_x/P_x, 1 \le j \le M_y/P_y$. If a process has neighbors, it should exchange the boundary data with its neighbors. The received boundary data are stored into the designed virtual boundaries. The lines 3–7 of Algorithm 1 are the preprocessing for the parallel algorithm. The lines 9–32 are the main time marching loops. T_1, T_2 are used to record the execution time.

4. Experimental Results and Discussion

The experiment platform is a cluster with distributed memory system (DSM) 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). For convenience to compare the runtime, the inner loop (lines 16–32) of Algorithm 1 is fixed as 3.

Table 1: Technical specifications of the experiment platform.

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

4.1. Numerical Example and Convergence of the Parallel Algorithm. The following time fractional ($\alpha = 0.4$) differential equation [4] was considered:

$$\frac{\partial^{0.4} u(x,y,t)}{\partial t^{0.4}} = \frac{2t^{1.6}}{\pi\Gamma(0.6)} \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{t^{1.6}}{12\pi\Gamma(0.6)} \frac{\partial^2 u(x,y,t)}{\partial y^2} + f(x,y,t),$$
$$u(x,y,0) = \sin(\pi x)\sin(\pi y), \quad (x,y) \in \Omega,$$
$$u(x,y,t)|_{\partial\Omega} = 0, \quad t \in [0,T], \tag{7}$$

where $f(x, y, t) = (25t^{1.6}/12\Gamma(0.6))(t^2 + 2)\sin(\pi x)\sin(\pi y)$, $\Omega = \{(x, y) \mid 0 < x < 1, 0 < y < 1\}$, and $\partial\Omega$ is the boundary of Ω. The exact solution of the above equation is $u(x, y, t) = (t^2 + 1)\sin(\pi x)\sin(\pi y)$.

The computational results for different α at $t = 1.0$ and $y = 0.5$ are shown in Figure 4. Figure 4 shows that the order of the fractional time derivative α governs the value of unknown u. With the increase of α to 1, (1) approaches the classical PDE. Figure 5 shows the numerical solutions with $\alpha = 0.4$, $t = 1.0$.

The parallel algorithm compares well with the exact analytic solution to the fractional partial differential equation in this test case of (7) with $\alpha = 0.4$, shown in Figure 6. The Δt and h are 1.0/100 and 1.0/10. The maximum absolute error is 8.36×10^{-3}.

4.2. Performance Improvement. For fixed $N = 10$, the performance comparison between single process and four processes (single CPU) is shown in Figure 7. The X step number in (6) is M, which is the x-coordinate of Figure 7. $M = M_x = M_y$ ranges from 2048 to 10240. With $M = 2028$, the runtime of one process is 23.45 seconds and the runtime of four processes is 6.64 seconds. The speedup is 3.53. With $M = 10240$, the runtime of one process is 803.88 seconds and the runtime of four processes is 192.76 seconds. The speedup is 4.17. From Figure 7, the parallel algorithm with fixed $N = 10$ is more than 4 times faster than the serial algorithm.

For fixed $M = 2560 = M_x = M_y$, the performance comparison between single process and four processes is shown in Figure 8. For single process, the X, Y step number is 2560. For four processes, the X, Y step number is 1280 with $P_x = 2$, $P_y = 2$. N ranges from 16 to 512. With $N = 16$, the runtime of one process is 17.63 seconds and the runtime

(1) init parallel environment
(2) **for** *all MPI processes* **do in parallel**
(3) get the input parameters like M_x, M_y, N, P_x, P_y, ϵ_0.
(4) allocate **local** memory u, c_1, c_2, f, Part1, v and so forth
(5) init variables and arrays
(6) get process id (p_x, p_y)
(7) compute the initial condition u^0 with $\phi(x, y)$ and boundary condition
(8) record time T_1
(9) **for** $n = 0$ **to** $N - 1$ **do**
(10) compute c_1, c_2, f et al.
(11) $v_{i,j} \leftarrow u^n_{i,j}$ with $I_{i,j}$
(12) $f_{i,j} \leftarrow f(x_i, y_j, (n+1)\tau)$ with $I_{i,j}$
(13) $\text{Part1}_{i,j} \leftarrow u^n_{i,j} + \tau^\alpha \Gamma(2-\alpha) f^{n+1}_{i,j}$ with $I_{i,j}$
(14) **for** $s = 1$ **to** n **do**
(15) $\text{Part1}_{i,j} \leftarrow \text{Part1}_{i,j} - b_s u^{n+1-s}_{i,j} + b_s u^{n-s}_{i,j}$ with $I_{i,j}$
(16) **while** $\epsilon \geq \epsilon_0$ **do**
(17) $u^{n+1}_{i,j} \leftarrow 1/(1 + 2c_1 + 2c_2)\,(c_1(v_{i+1,j} + v_{i-1,j}) + c_2(v_{i,j+1} + v_{i,j-1}))$ with $I_{i,j}$
(18) **if** $p_x < P_x$ **then**
(19) send right boundary to its right neighbor
(20) receive left boundary of its right neighbor
(21) **if** $p_y < P_y$ **then**
(22) send top boundary to its top neighbor
(23) receive bottom boundary of its top neighbor
(24) **if** $p_x > 0$ **then**
(25) send left boundary to its left neighbor
(26) receive right boundary of its left neighbor
(27) **if** $p_y < 0$ **then**
(28) send bottom boundary to its bottom neighbor
(29) receive top boundary of its bottom neighbor
(30) $\epsilon \leftarrow \max|v - u^{n+1}|$ with $I_{i,j}$
(31) get global maximum of ϵ of all processes
(32) $v_{i,j} \leftarrow u^{n+1}_{i,j}$ with $I_{i,j}$
(33) record time T_2
(34) output $T_2 - T_1$
(35) stop parallel environment

Algorithm 1: Parallel algorithm for 2D-TFDE.

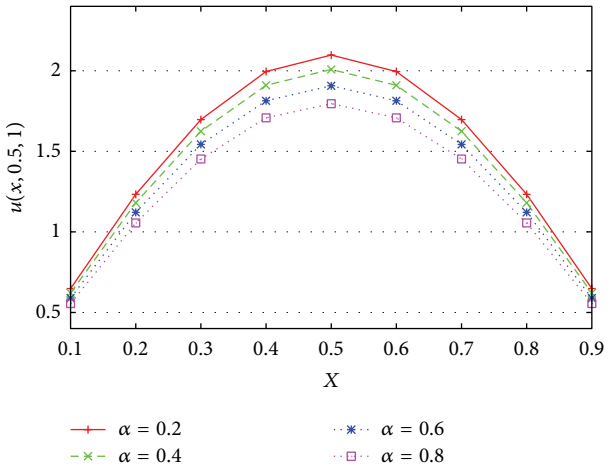

Figure 4: The numerical approximation whose transport is governed by the TFDE (7) for various $\alpha = 0.2, 0.4, 0.6, 0.8$ when $y = 0.5$, $t = 1.0$.

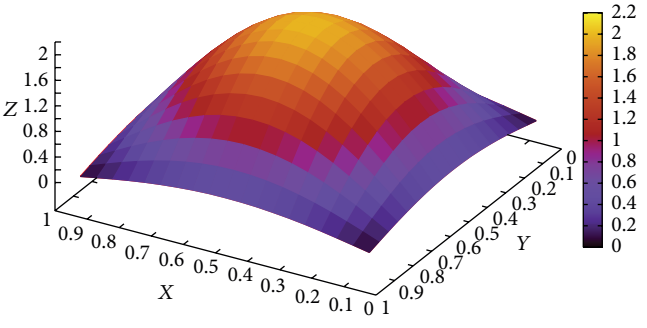

Figure 5: The approximation solution of (7) when $\alpha = 0.4$ and $t = 1.0$.

of four processes is 4.65 seconds. The speedup is 3.79. With $N = 512$, the runtime of one process is 4415.78 seconds and the runtime of four processes is 1394.99 seconds. The speedup

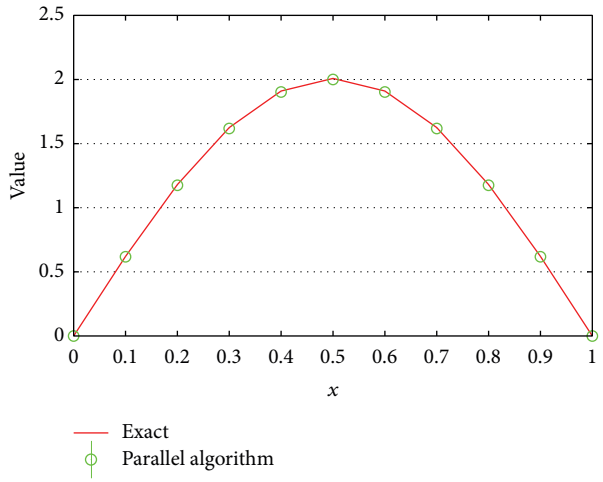

Figure 6: Comparison of exact solution to the solution of the parallel algorithm at time $t = 1.0$.

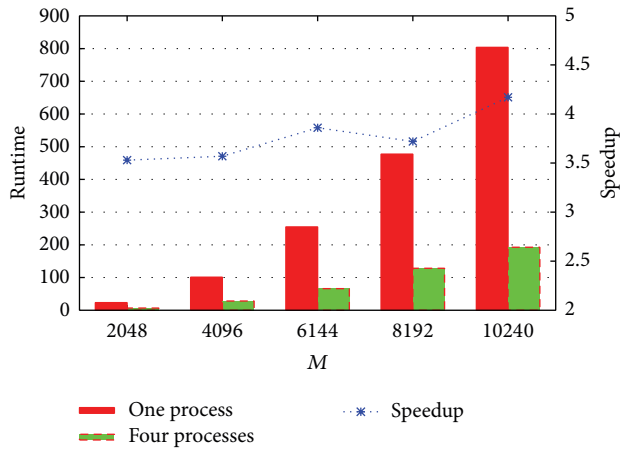

Figure 7: Performance comparison between one process and four processes on E5540 with fixed $N = 10$.

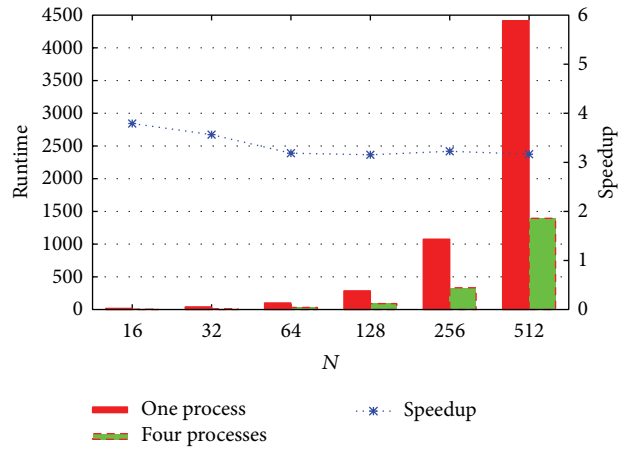

Figure 8: Performance comparison between one process and four processes on E5540 with fixed M.

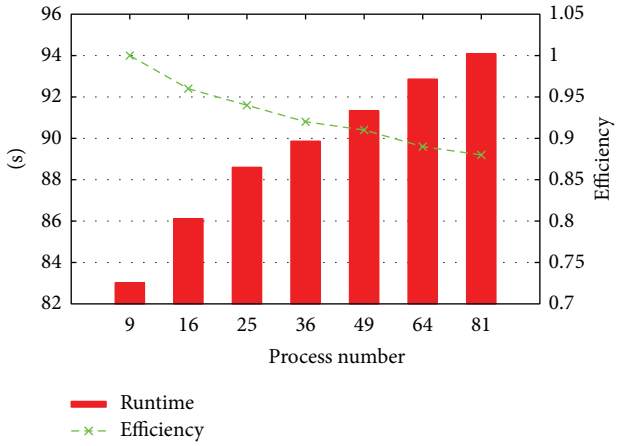

Figure 9: Scalability of the parallel algorithm on the cluster system.

is 3.16. The performance of four processes is about 3.2 times higher than the performance of single process with $M = 2560$.

4.3. Scalability. The scalability of the parallel algorithm on the large scale cluster system is shown in Figure 9. The technical specifications of the cluster system are listed in Table 1. N is fixed with 10 for all conditions. Each process has the same $(M_x/P_x, M_y/P_y)$ with $M = M_x = M_y$ and $P_x = P_y$. M varies from 16650, 33300, and 49950 for 9, 36, and 81 processes. The runtime of 9 processes is 83.02 seconds and the runtime of 81 processes is 94.08 seconds. The parallel efficiency of 81 processes is 88.24% compared with 9 processes. Here, the parallel efficiency is defined as the ratio of the runtime of different number of processes with the same work load on each process.

4.4. Discussion. The parallel Algorithm 1 will have good parallel scalability on distributed memory system. From Figure 3, we can see that each subdomain has only virtual boundary at every direction (top, bottom, left, and right). Assuming that the size of the subdomain is M_a, M_b $(M_a > 0, M_b > 0)$, the inner iteration of line 16 in Algorithm 1 has about $8M_aM_b$ arithmetic operations with $1/(1 + 2c_1 + 2c_2)$ precomputed. It needs to establish 8 communications for neighbors except the global communication for ϵ. The arithmetic operation of each time step besides the inner iteration is constant as KM_aM_b. K is bigger than $4nM_aM_b$. The communication data is $4M_a + 4M_b + 1$ grid point. Assuming that finishing one arithmetic operation needs time t_a and there are L inner iterations, the computing time of each time step is $(K + 8L)M_aM_b$. Assume that t_b is the time to establish the communication, t_c is the transform time for a grid point, and t_d is the global communication time. So, the total communication time for a time step is $L(9t_b + 4M_at_c + 4M_bt_c + t_d)$. The communication/computation ratio β is as follows:

$$\beta = \frac{L\left(9t_b + 4M_at_c + 4M_bt_c + t_d\right)}{(K + 8L)\, M_aM_b}. \tag{8}$$

Table 2: Impact of the source term on iteration times.

$f(x, y, t)$	$T = 2.0$	$T = 3.0$
$\frac{25t^{1.6}}{12\Gamma(0.6)}(t^2 + 2)\sin(\pi x)\sin(\pi y)$	284	444
$\frac{25}{12\Gamma(0.6)}2\sin(\pi x)\sin(\pi y)$	253	361
$\frac{25}{12\Gamma(0.6)}\sin(\pi x)\sin(\pi y)$	245	348
$\frac{1.0}{\Gamma(0.6)}\sin(\pi x)\sin(\pi y)$	238	336

The computation time is determined with the multiplication of $M_a M_b$ and the communication time is determined with the addition of M_a and M_b. The extreme of β is as follows:

$$\begin{aligned}
&\lim_{M_a, M_b \to \infty} \frac{L\left(9t_b + 4M_a t_c + 4M_b t_c + t_d\right)}{(K + 8L) M_a M_b} \\
&= \lim_{M_a \to \infty} \left(\lim_{M_b \to \infty} \frac{L\left(9t_b + 4M_a t_c + 4M_b t_c + t_d\right)}{(K + 8L) M_a M_b} \right) \\
&= \lim_{M_a \to \infty} \frac{L\left(4t_c\right)}{(K + 8L) M_a} = 0.
\end{aligned} \tag{9}$$

That means we can enhance the parallel efficiency by enlarging the size of subdomain.

The time t and number of grid points will affect the convergence property. The exact solution of (7) shows that $u(0.5, 0.5, t) = t^2 + 1$.

(1) The bigger t becomes, the more inner iterations are needed. With $M = M_x = M_y = 5$, $N = M^2$, the first inner time step t_1 needs 5 Jacobi iterations and the last inner time step t_N needs 31 iterations for $T = 1.0$. For $T = 2.0$, t_1 becomes 7 and t_N becomes 61.

(2) The bigger M becomes, the more inner iterations are needed. The T is fixed as 1.0. For $M = 10$, t_1 becomes 6 and t_N becomes 66. For $M = 10$, t_1 becomes 3 and t_N becomes 136.

The reason for the phenomenon above is that $\Delta u\ (u^{n+1} - u^n)$ changes dramatically if the source term $f(x, y, t)$ is big. The iteration times with $L = 1.0, M = 15, N = M^2$ are shown in Table 2.

The parallel algorithm is compatible with short memory principle [15]. The computing time $(K + 8L)M_a M_b$ will become small with a smaller K, which is determined by n. The Gauss-Seidel iteration method will have better convergent speed than Jacobi iteration method, but it is hard to parallelize the Gauss-Seidel method.

As analyzed in Section 3.1, the computational complexity is $O(M_x M_y N^2)$. Define the following function:

$$w = \log_2\left(\sqrt{T_2 - T_1}\right). \tag{10}$$

w varies almost linearly, as shown in Figure 10. Figure 10 shows that the heavy computation is a real challenge from the point of view of computer science.

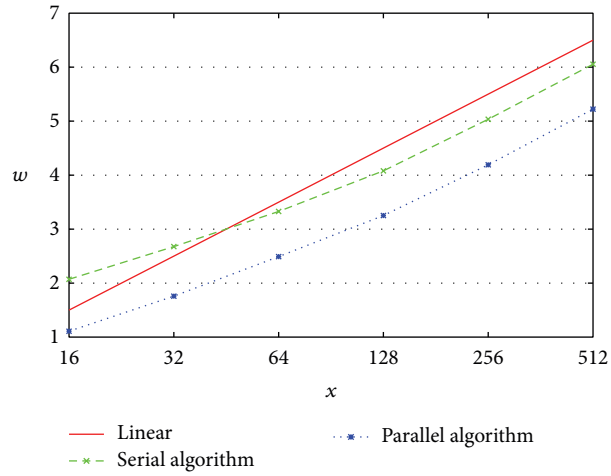

Figure 10: The linear variation of w.

The heavy memory usage is the other challenge besides the heavy computation. Ignoring the memory usage of the coefficients and the source term $f_{i,j}^n$, $u_{i,j}^n$ needs $8M_x M_y N$ bytes memory space. It needs 100 GB memory with $M_x = 10240$, $M_y = 10240$, and $N = 1024$. As discussed above, the bigger the M_x, M_y are, the smaller the β (communication/computation ratio) is. So, the heavy memory usage will limit the parallel efficiency of the parallel algorithm. This kind of contradictions exists in many places. One contradiction is the easy parallelization with bad convergence of the Jacobi iterative method. Another contradiction is the hard parallelization and good convergence of the Gauss-Seidel iterative method.

5. Conclusions and Future Work

In this paper, we present a parallel algorithm for 2D-TFDE with implicit differential method. The parallel solution is analyzed and implemented with MPI programming model. The experimental results show that the parallel algorithm compares well with the exact solution and can scale well on large scale distributed memory cluster system. So, the power of parallel computing for the time consuming fractional differential equations should be recognized.

The numerical solution for fractional equations is very computationally intensive. As a part of the future work, first, the numerical solution of high dimensional space fractional equations has global reliance on almost whole grid points, which is very challenging for real applications. Second, the Krylov subspace method with preconditioner will enhance the convergence for (4) and should be paid attention to. Third, accelerating the parallel algorithm on heterogeneous system [28] should be paid attention to.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research work is supported by the National Natural Science Foundation of China under Grant no. 11175253, also by 973 Program of China under Grant no. 61312701001. The authors would like to thank the anonymous reviewers for their helpful comments also.

References

[1] R. L. Magin, *Fractional Calculus in Bioengineering*, Begell House, Redding, Calif, USA, 2006.

[2] F. Liu, I. Turner, V. Anh, Q. Yang, and K. Burrage, "A numerical method for the fractional Fitzhugh-Nagumo monodomain model," *ANZIAM Journal*, vol. 54, pp. C608–C629, 2013.

[3] H. Ding and C. Li, "Mixed spline function method for reaction-diffusion equations," *Journal of Computational Physics*, vol. 242, pp. 103–123, 2013.

[4] P. Zhuang and F. Liu, "Finite difference approximation for two-dimensional time fractional diffusion equation," *Journal of Algorithms & Computational Technology*, vol. 1, no. 1, pp. 1–15, 2007.

[5] S. B. Yuste and L. Acedo, "An explicit finite difference method and a new von Neumann-type stability analysis for fractional diffusion equations," *SIAM Journal on Numerical Analysis*, vol. 42, no. 5, pp. 1862–1874, 2005.

[6] F. Liu, P. Zhuang, V. Anh, I. Turner, and K. Burrage, "Stability and convergence of the difference methods for the space-time fractional advection-diffusion equation," *Applied Mathematics and Computation*, vol. 191, no. 1, pp. 12–20, 2007.

[7] S. B. Yuste and J. Quintana-Murillo, "A finite difference method with non-uniform timesteps for fractional diffusion equations," *Computer Physics Communications*, vol. 183, no. 12, pp. 2594–2600, 2012.

[8] X. Zhang, P. Huang, X. Feng, and L. Wei, "Finite element method for two-dimensional time-fractional tricomi-type equations," *Numerical Methods for Partial Differential Equations*, vol. 29, no. 4, pp. 1081–1096, 2013.

[9] O. P. Agrawal, "A general finite element formulation for fractional variational problems," *Journal of Mathematical Analysis and Applications*, vol. 337, no. 1, pp. 1–12, 2008.

[10] C. Li, F. Zeng, and F. Liu, "Spectral approximations to the fractional integral and derivative," *Fractional Calculus and Applied Analysis*, vol. 15, no. 3, pp. 383–406, 2012.

[11] N. N. Leonenko, M. M. Meerschaert, and A. Sikorskii, "Fractional pearson diffusions," *Journal of Mathematical Analysis and Applications*, vol. 403, no. 2, pp. 532–546, 2013.

[12] P. Zhuang, Y. T. Gu, F. Liu, I. Turner, and P. K. D. V. Yarlagadda, "Time-dependent fractional advection-diffusion equations by an implicit MLS meshless method," *International Journal for Numerical Methods in Engineering*, vol. 88, no. 13, pp. 1346–1362, 2011.

[13] C. Tadjeran and M. M. Meerschaert, "A second-order accurate numerical method for the two-dimensional fractional diffusion equation," *Journal of Computational Physics*, vol. 220, no. 2, pp. 813–823, 2007.

[14] Y.-N. Zhang and Z.-Z. Sun, "Alternating direction implicit schemes for the two-dimensional fractional sub-diffusion equation," *Journal of Computational Physics*, vol. 230, no. 24, pp. 8713–8728, 2011.

[15] I. Podlubny, *Fractional Differential Equations*, Academic Press, San Diego, Calif, USA, 1999.

[16] C. Gong, W. Bao, and G. Tang, "A parallel algorithm for the Riesz fractional reaction-diffusion equation with explicit finite difference method," *Fractional Calculus and Applied Analysis*, vol. 16, no. 3, pp. 654–669, 2013.

[17] K. Diethelm, "An efficient parallel algorithm for the numerical solution of fractional differential equations," *Fractional Calculus and Applied Analysis*, vol. 14, no. 3, pp. 475–490, 2011.

[18] J. Yan, G.-M. Tan, and N.-H. Sun, "Optimizing parallel S_n sweeps on unstructured grids for multi-core clusters," *Journal of Computer Science and Technology*, vol. 28, no. 4, pp. 657–670, 2013.

[19] Z. Mo, A. Zhang, X. Cao et al., "JASMIN: a parallel software infrastructure for scientific computing," *Frontiers of Computer Science in China*, vol. 4, no. 4, pp. 480–488, 2010.

[20] F. Chen and J. Shen, "A GPU parallelized spectral method for elliptic equations in rectangular domains," *Journal of Computational Physics*, vol. 250, pp. 555–564, 2013.

[21] J. B. Haga, H. Osnes, and H. P. Langtangen, "A parallel block preconditioner for large-scale poroelasticity with highly heterogeneous material parameters," *Computational Geosciences*, vol. 16, no. 3, pp. 723–734, 2012.

[22] C. Gong, J. Liu, L. Chi, H. Huang, J. Fang, and Z. Gong, "GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method," *Journal of Computational Physics*, vol. 230, no. 15, pp. 6010–6022, 2011.

[23] A. Talamo, "Numerical solution of the time dependent neutron transport equation by the method of the characteristics," *Journal of Computational Physics*, vol. 240, pp. 248–267, 2013.

[24] C. Gong, J. Liu, H. Huang, and Z. Gong, "Particle transport with unstructured grid on GPU," *Computer Physics Communications*, vol. 183, no. 3, pp. 588–593, 2012.

[25] S. Pennycook, S. Hammond, S. Wright, J. Herdman, I. Miller, and S. Jarvis, "An investigation of the performance portability of OpenCL," *Journal of Parallel and Distributed Computing*, vol. 73, no. 11, pp. 1439–1450, 2012.

[26] J. Gu, X. Gu, and M. Gu, "A novel parallel quantum genetic algorithm for stochastic job shop scheduling," *Journal of Mathematical Analysis and Applications*, vol. 355, no. 1, pp. 63–81, 2009.

[27] F. Salvadore, M. Bernardini, and M. Botti, "GPU accelerated flow solver for direct numerical simulation of turbulent flows," *Journal of Computational Physics*, vol. 235, pp. 129–142, 2013.

[28] X.-J. Yang, X.-K. Liao, K. Lu, Q.-F. Hu, J.-Q. Song, and J.-S. Su, "The TianHe-1A supercomputer: its hardware and software," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 344–351, 2011.