Advances in Difference Equations
a SpringerOpen Journal

RESEARCH Open Access

An efficient parallel algorithm for Caputo fractional reaction-diffusion equation with implicit finite-difference method

Qinglin Wang[1,2*], Jie Liu[1,2], Chunye Gong[1,2], Xiantuo Tang[1,2], Guitao Fu[3] and Zuocheng Xing[1,2]

*Correspondence: wangqinglin.thu@gmail.com
[1]Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, 410073, China
[2]College of Computer, National University of Defense Technology, Changsha, 410073, China
Full list of author information is available at the end of the article

Abstract
An efficient parallel algorithm for Caputo fractional reaction-diffusion equation with implicit finite-difference method is proposed in this paper. The parallel algorithm consists of a parallel solver for linear tridiagonal equations and parallel vector arithmetic operations. For the parallel solver, in order to solve the linear tridiagonal equations efficiently, a new tridiagonal reduced system is developed with an elimination method. The experimental results show that the parallel algorithm is in good agreement with the analytic solution. The parallel implementation with 16 parallel processes on two eight-core Intel Xeon E5-2670 CPUs is 14.55 times faster than the serial one on single Xeon E5-2670 core.

MSC: Primary 34A08; 65Y05

Keywords: fractional reaction-diffusion equation; parallel computing; elimination method; tridiagonal reduced system

1 Introduction

Fractional differential equations (FDEs) refer to a class of differential equations which use derivatives of non-integer order [1, 2]. Fractional equations have proved to be very reliable models for many scientific and engineering problems [3, 4]. Because it is difficult to solve complex fractional problems analytically, more and more work focuses on numerical solutions [5, 6].

Recently, there has been great interest in FDEs [7–12]. Ahmad *et al.* [7] discussed the existence of the solution of a Caputo fractional reaction-diffusion equation with various boundary conditions. Rida *et al.* [8] applied the generalized differential transform method to solve nonlinear fractional reaction-diffusion partial differential equations. Chen *et al.* used the explicit finite-difference approximation [9] and implicit difference approximation [10] to solve the Riesz space fractional reaction-dispersion equation.

The numerical methods for FDEs include finite-difference methods [13, 14], finite element methods [15] and spectral methods [16–20]. Fractional reaction-diffusion equations are related to spatial and time coordinates, so the numerical solutions are often time-consuming. Large scale applications and algorithms in science and engineering such as neutron transport [21–23], computational fluid dynamics [24–26], large sparse systems [27] rely on parallel computing [24, 28, 29]. In order to overcome the difficulty, parallel

Springer

computing has been introduced into the numerical solutions for fractional equations [30–32]. Kai Diethelm [31] parallelized the fractional Adams-Bashforth-Moulton algorithm for the first time and the execution time of the algorithm was efficiently reduced. Gong *et al.* [33] presented a parallel solution for time fractional reaction-diffusion equation with explicit difference method. In the parallel solver, the technology named pre-computing fractional operator was used to optimize performance.

In this paper, we address an efficient parallel algorithm for time fractional reaction-diffusion equation with an implicit finite-difference method. In this parallel algorithm, the system of linear tridiagonal equations, vector-vector additions and constant-vector multiplications are efficiently processed in parallel. The linear tridiagonal system is parallelized with a new elimination method, which is effective and has simplicity in computer programming. The results indicate there is no significant difference between the implementation and the exact solution. The parallel algorithm with 16 parallel processes on two eight-core Intel Xeon E5-2670 CPUs runs 14.55 times faster than the serial algorithm.

This paper focuses on the Caputo fractional reaction-diffusion equation:

$$\begin{cases} {}_0^{\mathrm{C}}D_t^{\alpha}u(x,t) + \mu u(x,t) = \frac{\partial^2 u(x,t)}{\partial x^2} + Kf(x,t) \quad (0 < \alpha < 1), \\ u(x,0) = \phi(x), \quad x \in [0, x_R], \\ u(0,t) = u(x_R,t) = 0, \quad x \in [0,T], \end{cases} \tag{1}$$

on a finite domain $0 \leq x \leq x_R$ and $0 \leq t \leq T$, where μ and K are constants. If α equals 1, equation (1) is the classical reaction-diffusion equation. The fractional derivative is in the Caputo form.

2 Background

2.1 Numerical solution with implicit finite difference

The fractional derivative of $f(t)$ in the Caputo sense is defined as [1]

$${}_0^{\mathrm{C}}D_t^{\alpha}f(t) = \frac{1}{\Gamma(1-\alpha)} \int_0^t \frac{f'(\xi)}{(t-\xi)^{\alpha}}\, d\xi \quad (0 < \alpha < 1). \tag{2}$$

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

$$\begin{aligned} {}_0^{\mathrm{C}}D_t^{\alpha}f(t) &= \lim_{\xi \to 0, n\xi = t} \xi^{\alpha} \sum_{i=0}^{n} (-1)^i \binom{\alpha}{i} \\ &= \frac{f(0)t^{-\alpha}}{\Gamma(1-\alpha)} + \frac{1}{\Gamma(1-\alpha)} \int_0^t \frac{f'(\xi)}{(t-\xi)^{\alpha}}\, d\xi. \end{aligned} \tag{3}$$

Define $\tau = \frac{T}{N}$, $h = \frac{x_R}{M}$, $t_n = n\tau$, $x_i = 0 + ih$ for $0 \leq n \leq N$, $0 \leq i \leq M$. Define u_i^n, φ_i^n, and ϕ_i as the numerical approximations to $u(x_i, t_n)$, $f(x_i, t_n)$, and $\phi(x_i)$. We can get [11]

$${}_0^{\mathrm{C}}D_t^{\alpha}u(x,t)\big|_{x_i}^{t_n} = \frac{1}{\tau\Gamma(1-\alpha)} \left[b_0 u_i^n - \sum_{k=1}^{n-1} (b_{n-k-1} - b_{n-k}) u_i^k - b_{n-1} u_i^0 \right] + \bigcirc\left(\tau^{2-\alpha}\right), \tag{4}$$

where $1 \leq i \leq M-1$, $n \geq 1$, and

$$b_l = \frac{\tau^{1-\alpha}}{1-\alpha}\left[(l+1)^{1-\alpha} - l^{1-\alpha}\right], \quad l \geq 0. \tag{5}$$

Using implicit center difference scheme for $\frac{\partial^2 u(x,t)}{\partial x^2}$, we can get

$$\frac{\partial^2 u(x,t)}{\partial x^2}\bigg|_{x_i}^{t_n} = \frac{1}{h^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) + \bigcirc\left(h^2\right). \tag{6}$$

The implicit finite difference approximation [11] for equation (1) is

$$\begin{aligned}&\frac{1}{\tau\Gamma(1-\alpha)}\left[b_0 u_i^n - \sum_{k=1}^{n-1}(b_{n-k-1} - b_{n-k})u_i^k - b_{n-1}u_i^0\right] + \mu u_i^n \\ &\quad = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + K\varphi_i^n.\end{aligned} \tag{7}$$

Define $w = \tau\Gamma(1-\alpha)$, $\lambda_0 = b_0 + \mu w + 2w/h^2$, $\lambda_1 = -w/h^2$, $\sigma = Kw$, $U^n = (u_1^n, u_2^n, \ldots, u_{M-1}^n)^T$, $F^n = (\varphi_1^n, \varphi_2^n, \ldots, \varphi_{M-1}^n)^T$, and $r_l = b_l - b_{l+1}$. Equation (7) evolves as

$$AU^n = \sum_{k=1}^{n-1} r_{n-1-k}U^k + b_{n-1}U^0 + \sigma F^n, \tag{8}$$

where the matrix A is a tridiagonal matrix, defined as

$$A_{(M-1)\times(M-1)} = \begin{pmatrix} \lambda_0 & \lambda_1 & & & \\ \lambda_1 & \lambda_0 & \lambda_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \lambda_1 & \lambda_0 & \lambda_1 \\ & & & \lambda_1 & \lambda_0 \end{pmatrix}. \tag{9}$$

3 Parallel algorithm

3.1 Analysis

In order to get U^n from equation (8), two steps need to be processed. One step performs the right-sided computation of equation (8). Set $V^n = \sum_{k=1}^{n-1} r_{n-1-k}U^k + b_{n-1}U^0 + \sigma F^n$. The calculation of V^n mainly involves the constant-vector multiplications and vector-vector additions. The other step solves the tridiagonal linear equations $AU^n = V^n$. The constant-vector multiplications and vector-vector additions can easily be parallelized. So we only analyze the parallel implementation of solving the tridiagonal linear equations.

As $2|\lambda_1| < |\lambda_0|$, the tridiagonal matrix A is strictly diagonally dominant by rows. The dominance factor ϵ can be got by

$$\epsilon = \frac{|2\lambda_1|}{|\lambda_0|} = \frac{1}{\frac{Nx_R^2 b_0}{2M^2 T\Gamma(1-\alpha)} + \frac{x_R^2\mu}{2M^2} + 1}. \tag{10}$$

For strictly diagonally dominant linear systems, one parallel approximation algorithm has been proposed [34]. For given α and N, the dominance factor would be very close to one with big M. The approximation algorithm would decrease the precision of the solution and even exacerbate the convergence. In other words, applying the algorithm to this solution of tridiagonal linear equations with fixed M will obviously increase the number of time steps in order to keep the same precision. Thus, it is necessary to solve tridiagonal linear equations accurately. Moreover, each iteration on time step involves one system of tridiagonal linear equations. The right-hand side V^n varies while the tridiagonal matrix

A keeps constant in all time iterations. In order to avoid the repeated calculations, the transformation of the tridiagonal matrix is recorded in the parallel solution of tridiagonal linear equations.

3.2 Parallel solution of tridiagonal linear equations

Based on the analysis above, a parallel implementation of solving the tridiagonal linear equations is shown in Algorithm 1. The implementation is based on the divide-and-conquer principle. The number of parallel processes is p. Set $M - 1 = pk$ and $q = \frac{p}{2}$. The matrix A is divided into p blocks as follows:

$$A_{(M-1)\times(M-1)} = \begin{pmatrix} D_1 & C_1 & & & \\ G_2 & D_2 & C_2 & & \\ & \ddots & \ddots & \ddots & \\ & & G_{p-1} & D_{p-1} & C_{p-1} \\ & & & G_p & D_p \end{pmatrix}, \tag{11}$$

where

$$D_i = \begin{pmatrix} d_{(i-1)k+1} & c_{(i-1)k+1} & & & \\ g_{(i-1)k+2} & d_{(i-1)k+2} & c_{(i-1)k+2} & & \\ & \ddots & \ddots & \ddots & \\ & & g_{ik-1} & d_{ik-1} & c_{ik-1} \\ & & & g_{ik} & d_{ik} \end{pmatrix}, \quad i = 1, 2, \ldots, p, \tag{12}$$

$G_i = g_{(i-1)k+1} e_1 e_k^T$, $i = 2, 3, \ldots, p$, $C_j = c_{jk} e_k e_1^T$, $j = 1, 2, \ldots, p-1$. e_1 and e_k are k-dimensional unit vectors in which the first and last elements are one, respectively. Similarly, U and V can also have corresponding partitions. $U^n = (U_1, U_2, \ldots, U_P)^T$, $U_i = (u_{(i-1)k+1}, \ldots, u_{ik})^T$, and $V^n = (V_1, V_2, \ldots, V_P)^T$, $V_i = (v_{(i-1)k+1}, \ldots, v_{ik})^T$.

Line 3 allocates D_i, $g_{(i-1)k+1}$, c_{jk}, V_i to the ith process, $i = 1, \ldots, p$.

Lines 4-11 eliminate the lower and upper diagonal elements of sub-matrix D_i and transform diagonal elements to one. For the downward elimination, P_1 and P_p begin with the first row of D_i in line 9 while P_i $(1 < i < p)$ begins with the second row of D_i in line 5. After the downward elimination, the equations in all processes can be transformed to

$$P_i\ (i = 1): \quad \begin{pmatrix} 1 & c_1 & & & & \\ & 1 & c_2 & & & \\ & & \ddots & \ddots & & \\ & & & 1 & c_{k-1} & \\ & & & & 1 & c_k \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{k-1} \\ u_k \\ u_{k+1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{k-1} \\ v_k \end{pmatrix}, \tag{13}$$

$$P_i\ (1 < i < p): \quad \begin{pmatrix} g_{(i-1)k+1} & d_{(i-1)k+1} & c_{(i-1)k+1} & & & & \\ & g_{(i-1)k+2} & 1 & c_{(i-1)k+2} & & & \\ & \vdots & & \ddots & \ddots & & \\ & g_{ik-1} & & & 1 & c_{ik-1} & \\ & g_{ik} & & & & d_{ik} & c_{ik} \end{pmatrix}$$

Algorithm 1: Parallel solution for tridiagonal linear equations

```
   input  : $A$, $V$, $p$
   output: $U$
1  Initiate MPI and get $p$ (MPI Processes Size), $P_i$ (Current Process ID)
2  for all MPI processes do
3      Distribute task among MPI processes
4      if $P_i$ $(1 < i < p)$ then
5          elimit_down_mid_m()
6          elimit_up_mid_m()
7          transform_mid_v()
8      else
9          elimit_down_edg_m()
10         elimit_up_edg_m()
11         transform_edg_v()
       // Send equations to $P_1$ or $P_p$ and form the reduced
          system
12     com_form()
       // Solve the reduced system only with $P_1$ and $P_p$
13     solve_reduce()
       // Rec $u_{(i-1)k+1}$ and $u_{ik}$ from $P_1$ or $P_p$ and get the solution
          of the system
14     com_retrieve()
15 Output $T_2 - T_1$
```

$$\times \begin{pmatrix} u_{(i-1)k} \\ u_{(i-1)k+1} \\ u_{(i-1)k+2} \\ \vdots \\ u_{ik-1} \\ u_{ik} \\ u_{ik+1} \end{pmatrix} = \begin{pmatrix} v_{(i-1)k+1} \\ v_{(i-1)k+2} \\ \vdots \\ v_{ik-1} \\ v_{ik} \end{pmatrix}, \tag{14}$$

$$P_i\ (i = p):\quad \begin{pmatrix} g_{(p-1)k+1} & 1 & c_{(p-1)k+1} & & & \\ g_{(p-1)k+2} & & 1 & c_{(p-1)k+2} & & \\ \vdots & & & \ddots & \ddots & \\ g_{pk-1} & & & & 1 & c_{pk-1} \\ g_{pk} & & & & & 1 \end{pmatrix}$$

$$\times \begin{pmatrix} u_{(p-1)k} \\ u_{(p-1)k+1} \\ u_{(p-1)k+2} \\ \vdots \\ u_{pk-1} \\ u_{pk} \end{pmatrix} = \begin{pmatrix} v_{(p-1)k+1} \\ v_{(p-1)k+2} \\ \vdots \\ v_{pk-1} \\ v_{pk} \end{pmatrix}. \tag{15}$$

For the upward elimination, P_1 and P_p begin with the last row of D_i in line 10 while P_i $(1 < i < p)$ begins with the penultimate row of D_i in line 6. The upward elimination makes the equations in all processes become

$$P_i\ (i = 1):\quad \begin{pmatrix} 1 & & & & & c_1 \\ & 1 & & & & c_2 \\ & & \ddots & & & \vdots \\ & & & 1 & & c_{k-1} \\ & & & & 1 & c_k \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{k-1} \\ u_k \\ u_{k+1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{k-1} \\ v_k \end{pmatrix}, \tag{16}$$

$$P_i\ (1 < i < p):\quad \begin{pmatrix} g_{(i-1)k+1} & d_{(i-1)k+1} & & & c_{(i-1)k+1} & \\ & g_{(i-1)k+2} & 1 & & c_{(i-1)k+2} & \\ & \vdots & & \ddots & \vdots & \\ & g_{ik-1} & & 1 & c_{ik-1} & \\ & g_{ik} & & & d_{ik} & c_{ik} \end{pmatrix}$$

$$\times \begin{pmatrix} u_{(i-1)k} \\ u_{(i-1)k+1} \\ u_{(i-1)k+2} \\ \vdots \\ u_{ik-1} \\ u_{ik} \\ u_{ik+1} \end{pmatrix} = \begin{pmatrix} v_{(i-1)k+1} \\ v_{(i-1)k+2} \\ \vdots \\ v_{ik-1} \\ v_{ik} \end{pmatrix}, \tag{17}$$

$$P_i\ (i = p):\quad \begin{pmatrix} g_{(p-1)k+1} & 1 & & & \\ g_{(p-1)k+2} & & 1 & & \\ \vdots & & & \ddots & \\ g_{(p-1)k-1} & & & & 1 & \\ g_{pk} & & & & & 1 \end{pmatrix} \begin{pmatrix} u_{(p-1)k} \\ u_{(p-1)k+1} \\ u_{(p-1)k+2} \\ \vdots \\ u_{pk-1} \\ u_{pk} \end{pmatrix} = \begin{pmatrix} v_{(p-1)k+1} \\ v_{(p-1)k+2} \\ \vdots \\ v_{pk-1} \\ v_{pk} \end{pmatrix}. \tag{18}$$

In the elimination, the tridiagonal matrix and the right-hand-side vector are processed separately.

Line 12 means that P_i $(1 < i \leq q)$ and P_i $(q + 1 \leq i < p)$, respectively, sends the first and last equations to P_1 and P_p. Those equations, the last one in P_1 and the first one in P_p can form the reduced system of equations as follows:

$$\begin{pmatrix} 1 & c_k & & & & \\ g_{k+1} & d_{k+1} & c_{k+1} & & & \\ & g_{2k} & d_{2k} & c_{2k} & & \\ & & \ddots & \ddots & \ddots & \\ & & & g_{(p-2)k+1} & d_{(p-2)k+1} & c_{(p-2)k+1} \\ & & & g_{(p-1)k} & d_{(p-1)k} & c_{(p-1)k} \\ & & & & g_{(p-1)k+1} & 1 \end{pmatrix} \begin{pmatrix} u_k \\ u_{k+1} \\ u_{2k} \\ \vdots \\ u_{(p-2)k+1} \\ u_{(p-1)k} \\ u_{(p-1)k+1} \end{pmatrix} = \begin{pmatrix} v_k \\ v_{k+1} \\ v_{2k} \\ \vdots \\ v_{(p-2)k+1} \\ v_{(p-1)k} \\ v_{(p-1)k+1} \end{pmatrix}. \tag{19}$$

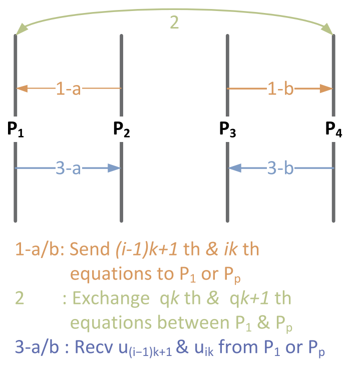
2
1-a
1-b
P1
P2
P3
P4
3-a
3-b
1-a/b: Send (i-1)k+1 th & ik th equations to P1 or Pp
2 : Exchange qk th & qk+1 th equations between P1 & Pp
3-a/b : Recv u(i-1)k+1 & uik from P1 or Pp

Figure 1 Communication in the parallel implementation of tridiagonal linear equations. The number of parallel processes is four.

We can find the coefficient matrix in equation (19) is still a tridiagonal matrix. Line 13 means the reduced system of equations is solved only with processes P_1 and P_p. P_1 and P_p deal with the reduced system of equations using the functions in lines 9-11 as well. P_1 and P_p exchange the qkth and $qk + 1$th equations after elimination, and both can obtain $u_{(q+1)k}$ and $u_{(q+1)k+1}$. In process P_1, $u_{(i-1)k+1}$ and u_{ik} are acquired and sent to P_i $(1 < i \leq q)$. In process P_p, $u_{(i-1)k+1}$ and u_{ik} are got and dispatched to P_i $(q + 1 \leq i < p)$. After P_i $(1 < i < p)$ receives $u_{(i-1)k+1}$ and u_{ik} from P_1 or P_p, U_i can be solved using equation (17). U_1 and U_p can also be figured out by equation (16) and equation (18), respectively.

For the parallel implementation of solving tridiagonal linear equations, there are totally three communications shown in Figure 1. The first communication means that P_i $(1 < i < p)$ sends the first and last equations to P_1 or P_p after the elimination. The second communication indicates that P_1 and P_p exchange the qkth and $qk+1$th equations in order to acquire $u_{(q+1)k}$ and $u_{(q+1)k+1}$ simultaneously in the solution of the reduced system. The third communication is that P_i $(1 < i < p)$ receives $u_{(i-1)k+1}$ and u_{ik} from P_1 or P_p after the reduced system is solved.

3.3 Implementation

The parallel algorithm for Caputo fractional reaction-diffusion equation with implicit finite-difference method is proposed as shown in Algorithm 2. $F(,)$, $U(,)$, $V()$, $A(,)$ are evenly distributed among all processes in order to void the load imbalance. Line 6 computes $F(,)$, $A()$, $r()$ and initializes $U(,)$, $V()$. Vector $r()$ is stored in all processes and calculated simultaneously. Lines 9-13 solve the tridiagonal linear equations $AU^0 = V$ for the first time. The loop of line 14 represents iterations on time steps. Since there are dependence between the solutions of adjacent time steps, the parallelization of the solution could be carried out only on space steps. In each iteration on time steps, line 16 solves $V^n = \sum_{k=1}^{n-1} r_{n-1-k} U^k + b_{n-1} U^0 + \sigma F^n$ in parallel, and mainly includes parallel constant-vector multiplications and vector-vector additions. Lines 18-22 solve the tridiagonal linear equations $AU^n = V^n$ for the $(n+1)$th time. As the tridiagonal matrix keeps constant in all time iterations, the solution of lines 18-22 does not deal with the tridiagonal matrix and only processes V^n with transform_mid_v() and transform_edg_v(). Particularly, the coefficient matrix of the reduced system also does not vary with time iterations. The elimination in line 22 simply involves the right-sided vector of the reduced system as well. Therefore, the communications in lines 18-22 only transfer the right-hand side of the corresponding equations rather than the entire ones.

Algorithm 2: Parallel solution for Caputo fractional reaction-diffusion equation with implicit finite-difference method

```
   input : $\alpha, M, N, L, T, f(x,t)$
1  $h \leftarrow L/M$, $\tau \leftarrow T/N$
2  Initiate MPI and get $p$ (MPI Processes Size), $P_i$ (Current Process ID)
3  for all MPI processes do
4      Distribute task among MPI processes
5      Declare local memory of $F(,)$, $U(,)$, $V()$, $A(,)$ in each process
6      Compute $F()$, $A()$, $r()$, and initialize $U(,)$, $V()$
7      Record time $T_1$
8      $V() \leftarrow K * F(:,0)$
       // solve $AU^0 = V^0$
9      if $P_i$ $(1 < i < p)$ then
10         elimit_down_mid_m(); elimit_up_mid_m(); transform_mid_v()
11     else
12         elimit_down_edg_m(); elimit_up_edg_m(); transform_edg_v()
13     com_form(); solve_reduce(); com_retrieve()
14     for $n = 1 \rightarrow N$ do
15         $V() \leftarrow K * F(:,n)$
16         for $j = 1 \rightarrow n$ do
17             $V() \leftarrow V() + U(:,j-1) * r(n-j)$
           // solve $AU^n = V^n$
18         if $P_i$ $(1 < i < p)$ then
19             transform_mid_v()
20         else
21             transform_edg_v()
22         com_form(); solve_reduce(); com_retrieve()
23     Record time $T_2$
24 Output $T_2 - T_1$
```

Table 1 The specifications of the experiment's platform

CPU	2 Intel Xeon E5-2670 CPUs, 8 cores/CPU, 2.6 GHz
Host OS	Linux Red Hat 4.4.5-6
Compiler Version	Intel v13.0.0
MPI Version	Intel v4.0.3

4 Experimental results and discussion

The experiment platform is a service node containing two eight-core Intel Xeon E5-2670 CPUs with 128 GB of memory. All codes are compiled with Intel C compiler with level-three optimization and run in double-precision floating-point. The specifications are listed in Table 1.

The following Caputo fractional reaction-diffusion equation [11] is considered, shown in equation (20):

$$\begin{cases} {}_0D_t^{\alpha} u(x,t) + \mu u(x,t) = \frac{\partial^2 u(x,t)}{\partial x^2} + Kf(x,t), \\ u(x,0) = 0, \quad x \in (0,2), \\ u(0,t) = u(2,t) = 0, \end{cases} \tag{20}$$

with $\mu = 1$, $K = 1$, $\alpha = 0.7$, and

$$f(x,t) = \frac{2}{\Gamma(2.3)} x(2-x)t^{1.3} + x(2-x)t^2 + 2t^2. \tag{21}$$

The exact solution of equation (20) is

$$u(x,t) = x(2-x)t^2. \tag{22}$$

4.1 Accuracy of parallel solution

For the example in equation (20), the parallel solution compares well with the exact analytic solution to the fractional partial differential equation shown in Figure 2. Δt and h are $\frac{1.0}{65}$ and $\frac{2.0}{65}$. The maximum absolute error is 7.47×10^{-4}. The difference between serial and parallel solution is only 4.22×10^{-15}. When different Δt and h are applied, the maximum errors between the exact analytic solution and the parallel solution are shown in Table 2. The maximum error gradually decreases with the increasing number of time and space steps. Altogether, our proposed parallel solution and the exact analytic solution have no noticeable artifacts.

4.2 Performance improvement

The performance comparison between serial solution (SS) and parallel solution (PS) is shown in Table 3 when different M and N are applied. In PS, 16 processes run in parallel. With $M = N = 65$, PS is a little slower than SS. When M and N are greater than or equal to 129, PS is faster than SS. Compared with SS, the speedup of PS increases gradually with

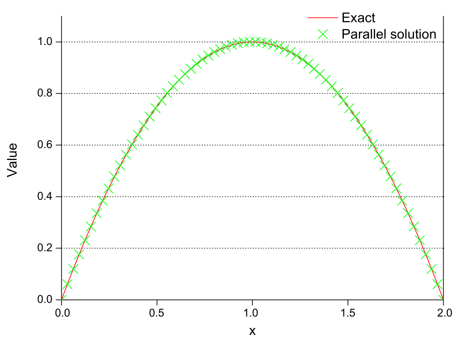

Figure 2 Result comparison between exact analytic solution with parallel solution at time $t = 1.0$ and $p = 16$, $M = N = 65$.

Table 2 Result comparison between the exact analytic solution and the parallel solution at time $t = 1.0$ and $p = 8$, where different M and N are applied

M* = *N	**65**	**129**	**257**	**513**	**1,025**	**2,049**
Max error	7.47E–04	3.07E–04	1.26E–04	5.12E–05	2.08E–05	8.47E–06

Table 3 Performance comparison between serial solution (SS) and parallel solution (PS) when different *M* and *N* are applied

M* = *N		**65**	**129**	**257**	**513**	**1,025**	**2,049**
Runtime (s)	SS	5.20E–04	3.31E–03	1.22E–02	7.35E–02	4.93E–01	4.03E+00
	PS	6.51E–04	1.42E–03	3.75E–03	1.15E–02	4.56E–02	2.77E–01
Speedup		0.80	2.34	3.26	6.37	10.80	14.55

In PS, the number of processes is set to 16.

Table 4 Performance of parallel solution with *M* = *N* = 2,049 when different numbers of processes are applied

No. of processes	**1**	**2**	**4**	**8**	**16**
Runtime (s)	4.03E+00	1.81E+00	9.50E–01	5.29E–01	2.77E–01
Speedup	1.00	2.22	4.24	7.62	14.55

increasing M and N. When M and N increase to 2,049, the runtime of SS reaches 4.03 seconds and that of PS is only 2.77×10^{-1} seconds. Therefore, the speedup between PS and SS rises to 14.55.

4.3 Scalability

With fixed $M = N = 2{,}049$, the performance comparison among the parallel solutions with a different number of processes is shown in Table 4. When two parallel processes are used, the speedup is 2.22. When four processes run in parallel, the speedup reaches 4.24. When more than one process is adopted, the total communication cost will increase with the number of parallel processes. However, the speedups in the two situations above outperform the corresponding perfect speedups. The main reason is that the solution of Caputo fractional reaction-diffusion equation with implicit finite-difference method is memory-intensive, and the decrease of memory overhead by the improvement of data locality is more than the increase of communication cost in the parallelization of two or four processes. When the number of processes grows to 16, the speedup is about 14.55 times and the scaling efficiency reaches 91%. Altogether, our proposed parallel solution has good scalability in performance.

5 Conclusions and future work

In this article, we propose a parallel algorithm for time fractional reaction-diffusion equation using the implicit finite-difference method. The algorithm includes a parallel solver for linear tridiagonal equations and parallel vector arithmetic operations. The solver is based on the divide-and-conquer principle and introduces a new tridiagonal reduced system with an elimination method. The experimental results shows the proposed parallel algorithm is valid and runs much more rapidly than the serial solution. The results also demonstrate the algorithm exhibits good scalability in performance. In addition, the introduced tridiagonal reduced system can be regarded as a general method for tridiagonal systems and applied on more applications. In the future, we would like to accelerate the solution of time fractional reaction-diffusion equation on heterogeneous architectures [35].

Competing interests
The authors declare that they have no competing interests.

Authors' contributions

Each of the authors contributed to each part of this work equally and read and approved the final version of the manuscript.

Author details

[1]Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, 410073, China. [2]College of Computer, National University of Defense Technology, Changsha, 410073, China. [3]Beijing Satellite Navigation Center, Beijing, 100094, China.

Acknowledgements

This research work is supported in part by the National Natural Science Foundation of China under Grant Nos. 11175253, 61303265, 61402039, 91430218, 91530324, 61303265, 61170083 and 61373032, in part by Specialized Research Fund for the Doctoral Program of Higher Education under Grant No. 20114307110001, in part by China Postdoctoral Science Foundation under Grant Nos. 2014M562570 and 2015T81127, and in part by 973 Program of China under Grant Nos. 61312701001 and 2014CB430205.

Received: 7 October 2015 Accepted: 28 July 2016 Published online: 11 August 2016

References

1. Podlubny, I: Fractional Differential Equations. Academic Press, San Diego (1999)
2. Debbouche, A, Baleanu, D: Controllability of fractional evolution nonlocal impulsive quasilinear delay integro-differential systems. Comput. Math. Appl. **62**(3), 1442-1450 (2011)
3. Lenzi, EK, Neto, RM, Tateishi, AA, Lenzi, MK, Ribeiro, HV: Fractional diffusion equations coupled by reaction terms. Phys. A, Stat. Mech. Appl. **458**, 9-16 (2016)
4. Hristov, J: Approximate solutions to time-fractional models by integral balance approach. In: Fractals and Fractional Dynamics, pp. 78-109 (2015)
5. Bhrawy, AH, Taha, TM, Alzahrani, EO, Baleanu, D, Alzahrani, AA: New operational matrices for solving fractional differential equations on the half-line. PLoS ONE **10**(5), e0126620 (2015)
6. Doha, EH, Bhrawy, AH, Baleanu, D, Ezz-Eldien, SS: On shifted Jacobi spectral approximations for solving fractional differential equations. Appl. Math. Comput. **219**(15), 8042-8056 (2013)
7. Ahmad, B, Alhothuali, MS, Alsulami, HH, Kirane, M, Timoshin, S: On a time fractional reaction diffusion equation. Appl. Math. Comput. **257**, 199-204 (2015)
8. Rida, SZ, El-Sayed, AMA, Arafa, AAM: On the solutions of time-fractional reaction-diffusion equations. Commun. Nonlinear Sci. Numer. Simul. **15**(12), 3847-3854 (2010). doi:10.1016/j.cnsns.2010.02.007
9. Chen, J, Liu, F, Turner, I, Anh, V: The fundamental and numerical solutions of the Riesz space fractional reaction-dispersion equation. ANZIAM J. **50**, 45-57 (2008)
10. Chen, J, Liu, F: Stability and convergence of an implicit difference approximation for the space Riesz fractional reaction-dispersion equation. Numer. Math. J. Chin. Univ., Engl. Ser. **16**(3), 253-264 (2007)
11. Chen, J: An implicit approximation for the Caputo fractional reaction-dispersion equation. J. Xiamen Univ. Natur. Sci. **46**(5), 616-619 (2007) (in Chinese)
12. Ding, X-L, Nieto, JJ: Analytical solutions for the multi-term time-space fractional reaction-diffusion equations on an infinite domain. Fract. Calc. Appl. Anal. **18**(3), 697-716 (2015)
13. Wang, H, Du, N: Fast alternating-direction finite difference methods for three-dimensional space-fractional diffusion equations. J. Comput. Phys. **258**, 305-318 (2014)
14. Ferrás, LL, Ford, NJ, Morgado, ML, Nóbrega, JM, Rebelo, MS: Fractional Pennes' bioheat equation: theoretical and numerical studies. Fract. Calc. Appl. Anal. **18**(4), 1080-1106 (2015)
15. Jiang, Y, Ma, J: High-order finite element methods for time-fractional partial differential equations. J. Comput. Appl. Math. **235**(11), 3285-3290 (2011)
16. Bhrawy, AH, Abdelkawy, MA, Alzahrani, AA, Baleanu, D, Alzahrani, EO: A Chebyshev-Laguerre-Gauss-Radau collocation scheme for solving a time fractional sub-diffusion equation on a semi-infinite domain. Proc. Rom. Acad., Ser. A **16**, 490-498 (2015)
17. Bhrawy, AH, Zaky, MA, Baleanu, D, Abdelkawy, MA: A novel spectral approximation for the two-dimensional fractional sub-diffusion problems. Rom. J. Phys. **60**(3-4), 344-359 (2015)
18. Bhrawy, AH, Doha, EH, Baleanu, D, Ezz-Eldien, SS: A spectral tau algorithm based on Jacobi operational matrix for numerical solution of time fractional diffusion-wave equations. J. Comput. Phys. **293**, 142-156 (2015)
19. Abdelkawy, MA, Zaky, MA, Bhrawy, AH, Baleanu, D: Numerical simulation of time variable fractional order mobile-immobile advection-dispersion model. Rom. Rep. Phys. **67**(3), 1-19 (2015)
20. Bhrawy, AH, Zaky, MA, Baleanu, D: New numerical approximations for space-time fractional Burgers' equations via a Legendre spectral-collocation method. Rom. Rep. Phys. **67**(2), 1-13 (2015)
21. Rosa, M, Warsa, JS, Perks, M: A cellwise block-Gauss-Seidel iterative method for multigroup S_N transport on a hybrid parallel computer architecture. Nucl. Sci. Eng. **173**(3), 209-226 (2013)
22. Gong, C, Liu, J, Chi, L, Huang, H, Fang, J, Gong, Z: GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method. J. Comput. Phys. **230**(15), 6010-6022 (2011). doi:10.1016/j.jcp.2011.04.010
23. Wang, Q, Liu, J, Gong, C, Xing, Z: Scalability of 3D deterministic particle transport on the Intel MIC architecture. Nucl. Sci. Tech. **26**(5), 50502 (2015)
24. Xu, C, Deng, X, Zhang, L, Fang, J, Wang, G, Jiang, Y, Cao, W, Che, Y, Wang, Y, Wang, Z, Liu, W, Cheng, X: Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer. J. Comput. Phys. **278**, 275-297 (2014)
25. Wang, Y-X, Zhang, L-L, Liu, W, Che, Y-G, Xu, C-F, Wang, Z-H, Zhuang, Y: Efficient parallel implementation of large scale 3D structured grid CFD applications on the Tianhe-1A supercomputer. Comput. Fluids **80**, 244-250 (2013)
26. Che, Y, Zhang, L, Xu, C, Wang, Y, Liu, W, Wang, Z: Optimization of a parallel CFD code and its performance evaluation on Tianhe-1A. Comput. Inform. **33**(6), 1377-1399 (2014)

27. Bai, Z-Z: Parallel multisplitting two-stage iterative methods for large sparse systems of weakly nonlinear equations. Numer. Algorithms **15**(3-4), 347-372 (1997). doi:10.1023/A:1019110324062
28. Mo, Z, Zhang, A, Cao, X, Liu, Q, Xu, X, An, H, Pei, W, Zhu, S: JASMIN: a parallel software infrastructure for scientific computing. Front. Comput. Sci. China **4**(4), 480-488 (2010). doi:10.1007/s11704-010-0120-5
29. Yang, B, Lu, K, Gao, Y, Wang, X, Xu, K: GPU acceleration of subgraph isomorphism search in large scale graph. J. Cent. South Univ. **22**, 2238-2249 (2015)
30. Gong, C, Bao, W, Tang, G, Jiang, Y, Liu, J: Computational challenge of fractional differential equations and the potential solutions: a survey. Math. Probl. Eng. **2015**, 258265 (2015)
31. Diethelm, K: An efficient parallel algorithm for the numerical solution of fractional differential equations. Fract. Calc. Appl. Anal. **14**, 475-490 (2011). doi:10.2478/s13540-011-0029-1
32. Gong, C, Bao, W, Tang, G: A parallel algorithm for the Riesz fractional reaction-diffusion equation with explicit finite difference method. Fract. Calc. Appl. Anal. **16**(3), 654-669 (2013)
33. Gong, C, Bao, W, Tang, G, Yang, B, Liu, J: An efficient parallel solution for Caputo fractional reaction-diffusion equation. J. Supercomput. **68**(3), 1521-1537 (2014). doi:10.1007/s11227-014-1123-z
34. Chi, L, Liu, J, Li, X: An effective parallel algorithm for tridiagonal linear equations. Chinese J. Comput. **22**(2), 218-221 (1999) (in Chinese)
35. Liao, X, Xiao, L, Yang, C, Lu, Y: MilkyWay-2 supercomputer: system and application. Front. Comput. Sci. **8**(3), 345-356 (2014)