Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2014, Article ID 820162, 7 pages
http://dx.doi.org/10.1155/2014/820162

Research Article

Solving the Caputo Fractional Reaction-Diffusion Equation on GPU

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

[1] *School of Computer Science, National University of Defense Technology, Changsha 410073, China*
[2] *Science and Technology on Space Physics Laboratory, Beijing 100076, China*
[3] *College of Aerospace Science and Engineering, 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 1 April 2014; Accepted 27 May 2014; Published 17 June 2014

Academic Editor: Dorian Popa

We present a parallel GPU solution of the Caputo fractional reaction-diffusion equation in one spatial dimension with explicit finite difference approximation. The parallel solution, which is implemented with CUDA programming model, consists of three procedures: preprocessing, parallel solver, and postprocessing. The parallel solver involves the parallel tridiagonal matrix vector multiplication, vector-vector addition, and constant vector multiplication. The most time consuming loop of vector-vector addition and constant vector multiplication is optimized and impressive performance improvement is got. The experimental results show that the GPU solution compares well with the exact solution. The optimized GPU solution on NVIDIA Quadro FX 5800 is 2.26 times faster than the optimized parallel CPU solution on multicore Intel Xeon E5540 CPU.

1. Introduction

The idea of fractional derivatives can be dated back to the 17th century. A fractional differential equation is a kind of equation which uses fractional derivatives. Fractional equations can be used to describe some physical phenomenons more accurately than the classical integer order differential equation [1]. The reaction-diffusion equations play an important role in dynamical systems of mathematics, physics, chemistry, bioinformatics, finance, and other research areas.

Some analytical methods were proposed for fractional differential equations [2, 3]. The stability of Cauchy fractional differential equations was studied [4, 5] and more attention should be paid to the interesting Ulam's type stability [6]. There have been a wide variety of numerical approximation methods proposed for fractional equations [7, 8], for example, finite difference method [9], finite element method, and spectral techniques [10]. Interest in fractional reaction-diffusion equations has increased [11]. In 2000, Henry and Wearne [12] derived a fractional reaction-diffusion equation from a continuous-time random walk model with temporal memory and sources. The fractional reaction-diffusion system with activator and inhibitor variables was studied by Gafiychuk et al. [13]. Haubold et al. [14] developed a solution in terms of the H-function for a unified reaction-diffusion equation. The generalized differential transform method [15] was presented for fractional reaction-diffusion equations. Saxena et al [16] gave investigation of a closed form solution of a generalized fractional reaction-diffusion equation.

Parallel computing is used to solve computation intensive applications simultaneously [17–19]. In recent years, the computing accelerators such as graphics processing unit (GPU) provided a new parallel method of accelerating computation intensive simulations [20–22]. The use of general purpose GPU is possible by the advance of programming models and hardware resources. The GPU programming models such as NVIDIA's compute unified device architecture (CUDA) [23] become more mature than before and simplify the development of nongraphic applications. GPU presents an energy efficient architecture for computation intensive domains like particle transport [24, 25] and molecular dynamics [26].

It is time consuming to numerically solve fractional differential equations for high spatial dimension or big time

integration. Short memory principle [27, 28] and parallel computing [29–32] can be used to overcome this difficulty. The parallel algorithms of one- and two- dimensional time fractional equations are studied and good parallel scalability is got [31, 32]. Optimization of the sum of constant vector multiplication is presented and 2-time speedup can be got [31]. The parallel implicit iterative algorithm was studied for two-dimensional time fractional problem at the first time [32].

Gong et al. [29] presented a parallel algorithm for Riesz space fractional equations. The parallel efficiency of the presented parallel algorithm of 64 processes is up to 79.39% compared with 8 processes on a distributed memory cluster system. Diethelm [30] implemented the fractional version of the second-order Adams-Bashforth-Moulton method for fractional ordinary equations on a parallel computer. Domain decomposition method is regarded as the basic mathematical background for many parallel applications. A domain decomposition algorithm for time fractional reaction-diffusion equation with implicit finite difference method was proposed [33]. The domain decomposition algorithm keeps the same parallelism but needs much fewer iterations, compared with Jacobi iteration in each time step, until nothing has been recorded on accelerating the numerical solution of Caputo fractional reaction-diffusion equation on GPU.

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

$$
\begin{aligned}
{}_0D_t^{\alpha} u(x,t) + \mu u(x,t) &= \frac{\partial^2 u(x,t)}{\partial x^2} + Kf(x,t) \quad (0 < \alpha < 1) \\
u(x,0) &= \phi(x), \quad x \in [0, x_R] \\
u(0,t) &= u(x_R,t) = 0, \quad x \in [0,T]
\end{aligned}
\tag{1}
$$

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

2. Background

2.1. Numerical Solution. The fractional derivative of $f(t)$ in the Caputo sense is defined as [27]

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

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

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

Define $\tau = T/N$, $h = x_R/(M+1)$, $t_n = n\tau$, and $x_i = 0 + ih$ for $0 \le n \le N$, $0 \le i \le M+1$. Define u_i^n, φ_i^n, and ϕ_i as the numerical approximation to $u(x_i, t_n)$, $f(x_i, t_n)$, and $\phi(x_i)$. We can get [11]

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

where $1 \le i \le M$, $n \ge 1$, and

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

Using explicit center difference scheme for $\partial^2 u(x,t)/\partial x^2$, we can get

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

The explicit finite difference approximation for (1) is

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

Define $s = b_0 + \mu\tau\Gamma(1-\alpha)$, $a_1 = \tau\Gamma(1-\alpha)/(sh^2)$, $a_2 = K\tau\Gamma(1-\alpha)/s$, $U^n = (u_1^n, u_2^n, \ldots, u_M^n)^T$, $F^n = (f_1^n, f_2^n, \ldots, f_M^n)^T$, $f_i^n = \tau\Gamma(1-\alpha)K\varphi_i^n$, and r_l as

$$
r_l = \frac{b_l - b_{l+1}}{s}. \tag{8}
$$

Equation (7) evolves as

$$
U^n = \sum_{k=1}^{n-1} r_{n-1-k} U^k + b_{n-1} U^0 + AU^{n-1} + a_2 F^n, \tag{9}
$$

where matrix A is a tridiagonal matrix, defined by

$$
A_{M\times M} = \begin{pmatrix}
-2a_1 & a_1 & & & \\
a_1 & -2a_1 & a_1 & & \\
 & \bullet & \bullet & & \\
 & & \bullet & \bullet & a_1 \\
 & & & a_1 & -2a_1
\end{pmatrix}. \tag{10}
$$

2.2. GPU Architecture and CUDA Programming Model. The architecture of GPU is optimized for rendering real-time graphics, a computation and memory access intensive problem domain with enormous inherent parallelism. Not like CPU, a much larger portion of GPUs resources is devoted to data processing rather than to caching or control flow. The NVIDIA Quadro FX 5800 GPU has 30 multiprocessor units

which are called the streaming multiprocessors (SMs). Each SM contains 8 SIMD CUDA cores. Each CUDA core runs at 1.30 GHz. The multiprocessors create, manage, and execute concurrent threads in hardware with near zero scheduling overhead. The single instruction multiple thread (SIMT) unit, which is akin to SIMD vector organizations, creates, manages, schedules, and executes threads in groups of 32 parallel threads called warp [23].

The programming model is a bridge between hardware and application. CUDA comes with a software environment that allows developers to use C as a high-level programming language. There are three key abstractions in CUDA: a hierarchy of thread execution model (grid or kernel, thread block, and thread), shared memory, and barrier synchronization. These abstractions provide fine-grained data level parallelism and thread parallelism, nested within coarse-grained data level parallelism and task level parallelism. Each CUDA kernel creates a single grid, which consists of many thread blocks. Each thread block schedules groups of threads that can share data through on-chip shared memory and synchronize with one another using barrier synchronization. Threads within a block are organized into warps which run in SIMT fashion. CUDA threads may access data from multiple memory spaces during their execution. The memory spaces include global, texture, and constant memory for grid, on-chip shared memory for thread block, and private register files for thread. The memory access time to different memory spaces varies from several to hundreds of cycles. These memory accesses must be coalesced to improve the performance of global memory access.

3. Details of GPU Solution

The parallel solution consists of three parts. The first part is preprocessing, which prepares the initial matrices, vectors, and so on. The second part is the parallel solver, which focuses on the iteration of time step with (9). The third part is postprocessing, which outputs the final results and so on.

The preprocessing includes initialization of parallel environment, distribution of computing task, allocation of local memory space, and initialization of variables and arrays. Matrices $A_{M\times M}$ and $F_{M\times N}$ are prepared before the computation of (9). For example, matrix A can be got according to (10). The postprocessing is simple. The results of 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 plot. Other operations of postprocessing include free memory space and stop the parallel environment.

In order to get U^n, the right-sided computation of (9) should be performed. There are mainly one tridiagonal matrix vector multiplication, many constant vector multiplications, and many vector-vector additions in the right-sided computation.

(1) The tridiagonal matrix vector multiplication is AU^n.

```
(1)  Init CUDA environment
(2)  Allocate GPU global memory A, U, F, r, b ....
(3)  Init variables and arrays on GPU
(4)  record time T_1
(5)  call kernel initU^0 ⟨⟨⟨M/BS, BS⟩⟩⟩ (...)
(6)  for n = 1 to N by Step 1 do
(7)      call kernel trimvmCU ⟨⟨⟨M/BS, BS⟩⟩⟩ (...)
(8)      call kernel cvm_vvaCU ⟨⟨⟨M/BS, BS⟩⟩⟩ (...)
(9)      for k = 1 to n by Step 1 do
(10)         call kernel cvmaCU ⟨⟨⟨M/BS, BS⟩⟩⟩ (...)
(11)     call kernel cvmCU ⟨⟨⟨M/BS, BS⟩⟩⟩ (...)
(12) record time T_2
(13) output T_2 − T_1 and U^N ...
(14) free GPU memory and stop CUDA environment
```

Algorithm 1: Parallel solution for Caputo fractional reaction-diffusion equation with CUDA.

(2) The constant vector multiplications are $V^k = r_{n-1-k}U^k$, a_2F^n, and so on.

(3) The vector-vector additions are $\sum_{k=1}^{n-1} V^k$ and so on.

The parallel solution uses the data level parallelism of GPU architecture. The parallel solution with CUDA for (1) is described in Algorithm 1. The preprocessing involves lines 1 to 3. The parallel solver, which is the most time consuming procedure, involves lines 4 to 12. The postprocessing involves lines 13 to 14 and other additional operations are not shown in Algorithm 1. Because the time spent on the preprocessing and postprocessing is trivial when the number of time steps is big enough, the preprocessing and postprocessing time is omitted for the measured time. T_1 and T_2 are used to record the measured time of the parallel CPU and GPU solutions.

The parallel solution uses the data level parallelism of GPU architecture. The parallel solution with CUDA for (1) is described in Algorithm 1. The preprocessing involves lines 1 to 3. The parallel solver, which is the most time consuming procedure, involves lines 4 to 12. The postprocessing involves lines 13 to 14 and other additional operations are not shown in Algorithm 1. BS stands for the CUDA thread block size and M/BS is the number of CUDA thread blocks. BS is the predefined constant like 16, 32, 64, and so forth. Because the time spent on the preprocessing and postprocessing is trivial when the number of time steps is big enough, the preprocessing and postprocessing time is omitted for the measured time. T_1 and T_2 are used to record the measured time of the serial and parallel solution.

Except for the initialization of variables and arrays, there are four CUDA kernels. The first kernel is $initU^0$, which computes the initial condition according to $\phi(x)$ in (1). The second kernel is *trimvmCU*, which performs the tridiagonal matrix vector multiplication. The third kernel is *cvm_vvaCU*, which stands for constant vector multiplication and vector-vector addition. The fourth kernel is *cvmaCU*, which performs the constant vector multiplication and vector-vector addition. The last kernel is *cvmCU*, which stands for constant

```
input: colOut, colIn, a_2, U
output: U
(1) for all threads in each thread block do in parallel
(2)     local variable gid, ξ
(3)     gid ← threadId.x + blockId.x · blockDim.x
(4)     ξ ← a_2 × U[M · colIn + gid]
(5)     U[M · colOut + gid] ← U[M · colOut + gid] + ξ
```

Algorithm 2: CUDA kernel for constant vector multiplication and vector vector addition.

vector multiplication. The CUDA kernels $initU^1$ and *cvmCU* are simple and will not be described in detail.

3.1. Implementation. The CUDA kernel *cvmaCU* for constant vector multiplication and vector-vector addition is shown in Algorithm 2. Algorithm 2 computes $U^{colOut} + a_2U^{colIn}$ and saves the final vector into U^{colOut}.

Most elements of tridiagonal matrix A are zero. The most common data structure used to store a sparse matrix for sparse matrix vector multiplication computations is compressed sparse row (CSR) format [34] shown in

$$A_{3\times M} = \begin{pmatrix} 0 & a_1 & \cdots & a_1 & a_1 \\ -2a_1 & -2a_1 & \cdots & -2a_1 & -2a_1 \\ a_1 & a_1 & \cdots & a_1 & 0 \end{pmatrix}. \tag{11}$$

So in the following parts of this paper, matrix A stands for the format of (11) not the format of (10). With the format of (11), the global memory is coalesced and can improve the performance of global memory access.

The CUDA kernel *trimvmCU* for tridiagonal matrix vector multiplication is shown in Algorithm 3. One thread block deals with the multiplication of one row of matrix A and one column of U. Algorithm 3 computes AU^{n-1} and saves the final vector into U^n. The shared memory is used to improve the memory access speed. The synchronization function *__syncthreads*() is used to ensure the correctness of the logic of the algorithm.

In Algorithm 3, each GPU thread deals with the multiplication of one row of tridiagonal matrix A and vector U^{n-1}. Each thread needs to use three elements of vector U^n. The nearest two threads use the same two elements of vector U^{n-1}. We can use the shared memory to improve the memory access performance. So the elements of vector U^{n-1} which will be used by threads in a thread block are stored into shared memory, shown between lines 6 and 16 of Algorithm 3. The real computation of tridiagonal matrix multiplication is shown between lines 18 and 21. Finally, the results are stored into the global memory of U^n.

3.2. Optimization. In Algorithm 1, the kernels *cvm_vvaCU*, *cvmCU*, and *trimvmCU* are invoked N times. In each time step, kernel *cvmaCU* is invoked n $(1, 2, \ldots, N)$ times. Because

$$\frac{I(I-1)}{2} = 1 + 2 + \cdots I \tag{12}$$

```
input: n, M, BS, A, U
output: U
(1)  shared memory sm[BS + 2]
(2)  for all threads in every thread block do in parallel
(3)      local variables tid, gid, BS, ξ
(4)      tid ← threadId.x
(5)      gid ← threadId.x + blockId.x · blockDim.x
(6)      sm[tid + 1] ← U[M · (n − 1) + gid]
(7)      if 0 == tid then
(8)          if 0 == gid then
(9)              sm[0] ← 0
(10)         else
(11)             sm[0] ← U[M · (n − 1) + gid − 1]
(12)     if BS − 1 == tid then
(13)         if M == gid then
(14)             sm[BS + 1] ← 0
(15)         else
(16)             sm[BS + 1] ← U[M · (n − 1) + gid + 1]
(17)     __syncthreads()
(18)     ξ ← 0.0
(19)     ξ ← ξ + sm[tid + 0] · A[0 ∗ M + gid]
(20)     ξ ← ξ + sm[tid + 1] · A[1 ∗ M + gid]
(21)     ξ ← ξ + sm[tid + 2] · A[2 ∗ M + gid]
(22)     U[M · n + gid] ← U[M · n + gid] + ξ
```

Algorithm 3: CUDA kernel for tridiagonal matrix vector multiplication.

```
input: n, gpuR, U
output: U
(1) for all threads in every thread block do in parallel
(2)     local variables gid, ξ, k
(3)     gid ← threadId.x + blockId.x · blockDim.x
(4)     ξ ← 0
(5)     for k = 1 to n − 1 by Step 1 do
(6)         ξ ← ξ + gpuR[k] · U[M · (n − 1 − k) + gid]
(7)     U[M · n + gid] ← U[M · n + gid] + ξ
```

Algorithm 4: Optimized CUDA kernel for constant vector multiplication and vector-vector addition.

the total number of the invocations of kernel *cvmaCU* is $N^2(N-1)/2$. The most time consuming part of Algorithm 1 is the loop of line 9. Loop 9 can be combined into one CUDA kernel *cvmaOptCU* as shown in Algorithm 4. The array *gpuR* is the coefficient of (8) in global memory.

So the optimized parallel solution for Caputo fractional reaction-diffusion equation is similar to Algorithm 1 except that the loop (lines 9-10) in Algorithm 1 is replaced with the optimized CUDA kernel of Algorithm 4.

4. Experimental Results

4.1. Experiment Platforms. The experiment platforms consist of one GPU and one CPU listed in Table 1. For the purpose of fair comparisons, we measure the performance provided by GPU compared to the MPI code running on multicore

Table 1: Technical specifications of experiment platforms.

CPU	Intel Xeon E5540, 4 cores, 2.53 GHz
GPU	NVIDIA Quadro FX 5800, 240 SPs, 1.30 GHz
Operating system	Kylin server version 3.1
CPU compiler GPU compiler	mpif90, Intel Fortran version 11.1 NVCC, CDUA version 3.0
Communication	MPICH2 version 1.3rc2

CPU [31]. Both codes run on double precision floating point operations. The CPU code is compiled by the mpif90 compiler with level three optimization. The GPU code is compiled by the NVCC compiler provided by CUDA version 3.0 with level three optimization too.

4.2. Numerical Example. The following Caputo fractional reaction-diffusion equation [11] was considered:

$$
\begin{aligned}
{}_0D_t^{\alpha} u(x,t) + \mu u(x,t) &= \frac{\partial^2 u(x,t)}{\partial x^2} + Kf(x,t) \quad (0 < \alpha < 1) \\
u(x,0) &= 0, \quad x \in (0,2) \\
u(0,t) &= u(2,t) = 0
\end{aligned}
\tag{13}
$$

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

$$
f(x,t) = \frac{2}{\Gamma(2.3)} x(2-x)t^{1.3} + x(2-x)t^2 + 2t^2. \tag{14}
$$

The exact solution of (13) is

$$
u(x,t) = x(2-x)t^2. \tag{15}
$$

4.3. Accuracy of the GPU Implementation. The GPU solution compares well with the exact solution to the time fractional diffusion equation in this test case of (13), shown in Figure 1. The τ and h for the GPU solution are $T/2048$ and $2.0/16$. The maximum absolute errors for $T = 0.3$, 0.5, and 0.7 are 3.29×10^{-5}, 1.07×10^{-4}, and 2.30×10^{-4}. In fact, the accuracy and convergence of the GPU solution are the same as the serial and parallel MPI solution on CPU [31].

4.4. Total Performance Improvement. In this section, the performance of the optimized GPU solution presented in this paper is compared with the performance of the parallel CPU solution [31]. The parallel CPU solution makes full use of four cores of E5540. The optimized GPU solution is presented in Section 3.2.

For fixed $N = 128$, the performance comparison between GPU and multicore CPU is shown in Table 2. The thread block size is 32.

For fixed $M = 122880$, the performance comparison between optimized GPU solution and parallel CPU solution is shown in Table 3. The thread block size is 32.

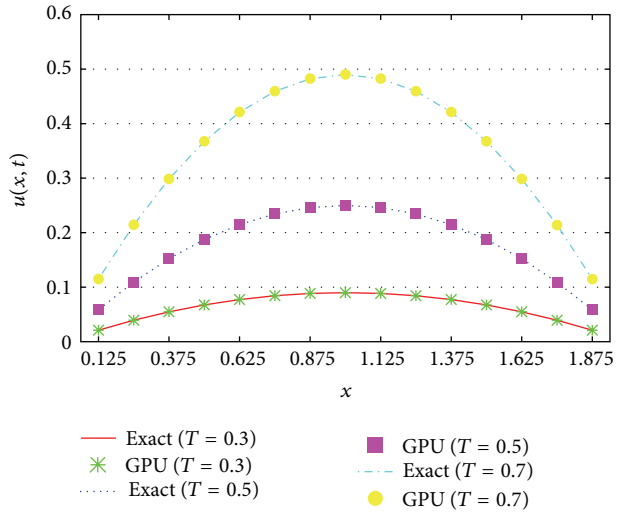

Figure 1: Comparison of exact solution to the parallel GPU solution at $T = 0.3, 0.5, 0.7$.

Table 2: Performance comparison between optimized GPU solution on Quadro FX 5800 and parallel CPU solution on E5540 with fixed $N = 128$.

M	CPU	GPU	Speedup
245760	0.58	0.46	1.26
491520	1.23	0.90	1.36
737280	1.95	1.27	1.54
983040	3.27	1.89	1.73
1228800	4.66	2.06	2.26

Table 3: Performance comparison between optimized GPU solution on Quadro FX 5800 and parallel CPU solution on E5540 with fixed $M = 122880$.

N	CPU	GPU	Speedup
128	0.29	0.25	1.14
256	1.06	0.84	1.26
512	3.96	2.82	1.40
1024	15.20	9.33	1.63
2048	60.16	33.59	1.79

4.5. Performance Issues of GPU Solution. With $M = 491520$, $N = 128$, and thread block size 64, the runtime of Algorithm 1 on Quadro FX 5800 is 1.228 seconds. Without the loop of line 9 in Algorithm 1, the runtime is only 0.032 seconds. That is to say, about 97.39% of runtime is spent on the loop of line 9. This is the reason why we develop the optimized GPU solution with an optimized CUDA kernel of Algorithm 4.

The impact of the optimized CUDA kernel on constant vector multiplication and vector-vector addition with fixed $N = 128$ is shown in Table 4. The performance improvement with fixed $M = 491520$ is shown in Table 5. All thread block sizes are 64. The basic GPU solution is Algorithm 1 and the optimized GPU solution uses the optimized CUDA kernel of Algorithm 4.

Table 4: Performance improvement for fixed N with the optimization of constant vector multiplication and vector-vector addition.

M	Original	Optimization	Speedup
245760	0.64	0.53	1.21
491520	1.26	1.05	1.20
737280	1.85	1.56	1.19
983040	2.53	2.10	1.20
1228800	3.06	2.59	1.18

Table 5: Performance improvement for fixed M with the optimization of constant vector multiplication and vector-vector addition.

N	Original	Optimization	Speedup
256	0.72	0.53	1.36
512	2.86	1.75	1.63
1024	11.36	5.60	2.03
2048	45.32	18.65	2.43
4096	181.01	66.71	2.71

Table 6: Impact of thread block size (BS).

BS	Runtime	BS	Runtime
4	13.5	64	3.35
8	6.84	128	3.61
16	3.59	256	3.55
32	2.82	512	3.47

The thread block size (BS) is a key parameter for parallel GPU algorithms. The impact of BS is shown in Table 6. From Table 6, we can see that thread block size 32 is the best choice.

5. Conclusions and Future Work

In this paper, the numerical solution of Caputo fractional reaction-diffusion equation with explicit finite difference method is accelerated on GPU. The iteration of time step consists of tridiagonal matrix vector multiplication, constant vector multiplication, and vector-vector addition. The details of the GPU solution and some basic CUDA kernels are presented. The most time consuming loop (constant vector multiplication and vector-vector addition) is optimized. The experimental results show the GPU solution compares well with the exact analytic solution and is much faster than parallel CPU solution. So the power of parallel computing on GPU for solving fractional applications should be recognized.

As a part of the future work, first, the stability, like Ulam's type, of different fractional equations should be paid attention to [35–37]. Second, parallelizing the implicit numerical method of fractional differential equations is challenging.

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 in part by the National Natural Science Foundation of China under Grants no. 11175253 and no. 61170083, in part by Specialized Research Fund for the Doctoral Program of Higher Education under Grant no. 20114307110001, and in part by 973 Program of China under Grant no. 61312701001. The authors would like to thank the anonymous reviewers for their helpful comments as well.

References

[1] F. Huang and F. Liu, "The time fractional diffusion equation and the advection-dispersion equation," *The ANZIAM Journal*, vol. 46, no. 3, pp. 317–330, 2005.

[2] S. Chen and X. Jiang, "Analytical solutions to time-fractional partial differential equations in a two-dimensional multilayer annulus," *Physica A: Statistical Mechanics and Its Applications*, vol. 391, no. 15, pp. 3865–3874, 2012.

[3] R. K. Pandey, O. P. Singh, and V. K. Baranwal, "An analytic algorithm for the space-time fractional advection-dispersion equation," *Computer Physics Communications*, vol. 182, no. 5, pp. 1134–1144, 2011.

[4] R. W. Ibrahim, "Ulam-hyers stability for cauchy fractional differential equation in the unit disk," *Abstract and Applied Analysis*, vol. 2012, Article ID 613270, 10 pages, 2012.

[5] J. Brzdęk, N. Brillouët-Belluot, K. Ciepliński, and B. Xu, "Ulam's type stability," *Abstract and Applied Analysis*, vol. 2012, Article ID 329702, 2 pages, 2012.

[6] N. Brillouët-Belluot, J. Brzdk, and K. Ciepliński, "On some recent developments in ulam's type stability," *Abstract and Applied Analysis*, vol. 2012, Article ID 716936, 41 pages, 2012.

[7] Q. Liu, F. Liu, I. Turner, and V. Anh, "Numerical simulation for the 3D seepage flow with fractional derivatives in porous media," *IMA Journal of Applied Mathematics*, vol. 74, no. 2, pp. 201–229, 2009.

[8] A. Ashyralyev and Z. Cakir, "On the numerical solution of fractional parabolic partial differential equations with the dirichlet condition," *Discrete Dynamics in Nature and Society*, vol. 2012, Article ID 696179, 15 pages, 2012.

[9] A. Ashyralyev and F. Dal, "Finite difference and iteration methods for fractional hyperbolic partial differential equations with the neumann condition," *Discrete Dynamics in Nature and Society*, vol. 2012, Article ID 434976, 15 pages, 2012.

[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] J. H. Chen, "An implicit approximation for the caputo fractional reaction-dispersion equation," *Journal of Xiamen University (Natural Science)*, vol. 46, no. 5, pp. 616–619, 2007 (Chinese).

[12] B. I. Henry and S. L. Wearne, "Fractional reaction-diffusion," *Physica A: Statistical Mechanics and Its Applications*, vol. 276, no. 3-4, pp. 448–455, 2000.

[13] V. Gafiychuk, B. Datsko, and V. Meleshko, "Mathematical modeling of time fractional reaction-diffusion systems," *Journal of Computational and Applied Mathematics*, vol. 220, no. 1-2, pp. 215–225, 2008.

[14] H. J. Haubold, A. M. Mathai, and R. K. Saxena, "Further solutions of fractional reaction-diffusion equations in terms of

the h-function," *Journal of Computational and Applied Mathematics*, vol. 235, no. 5, pp. 1311–1316, 2011.

[15] S. Z. Rida, A. M. A. El-Sayed, and A. A. M. Arafa, "On the solutions of time-fractional reaction-diffusion equations," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 12, pp. 3847–3854, 2010.

[16] R. K. Saxena, A. M. Mathai, and H. J. Haubold, "Solution of generalized fractional reaction-diffusion equations," *Astrophysics and Space Science*, vol. 305, no. 3, pp. 305–313, 2006.

[17] S. J. Pennycook, S. D. Hammond, G. R. Mudalige, S. A. Wright, and S. A. Jarvis, "On the acceleration of wavefront applications using distributed many-core architectures," *The Computer Journal*, vol. 55, no. 2, pp. 138–153, 2012.

[18] 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.

[19] R. Zhang, "A three-stage optimization algorithm for the stochastic parallel machine scheduling problem with adjustable production rates," *Discrete Dynamics in Nature and Society*, vol. 2013, Article ID 280560, 15 pages, 2013.

[20] 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.

[21] Y.-X. Wang, L.-L. Zhang, W. Liu et al., "Efficient parallel implementation of large scale 3D structured grid CFD applications on the Tianhe-1A supercomputer," *Computers and Fluids*, vol. 80, pp. 244–250, 2013.

[22] C. Xu, X. Deng, L. Zhang et al., "Parallelizing a high-order CFD software for 3D, multi-block, structural grids on the TianHe-1A supercomputer," in *Supercomputing*, J. Kunkel, T. Ludwig, and H. Meuer, Eds., vol. 7905 of *Lecture Notes in Computer Science*, pp. 26–39, Springer, Heidelberg, Germany, 2013.

[23] NVIDIA Corporation, *CUDA Programming Guide Version 3.1*, 2010.

[24] 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.

[25] 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.

[26] Q. Wu, C. Yang, T. Tang, and L. Xiao, "Exploiting hierarchy parallelism for molecular dynamics on a petascale heterogeneous system," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1592–1604, 2013.

[27] I. Podlubny, *Fractional Differential Equations*, vol. 198 of *Mathematics in Science and Engineering*, Academic Press, San Diego, Calif, USA, 1999.

[28] Y. Xu and Z. He, "The short memory principle for solving abel differential equation of fractional order," *Computers & Mathematics with Applications*, vol. 62, no. 12, pp. 4796–4805, 2011.

[29] 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.

[30] 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.

[31] C. Gong, W. Bao, G. Tang, B. Yang, and J. Liu, "An efficient parallel solution for Caputo fractional reaction-diffusion equation," *The Journal of Supercomputing*, 2014.

[32] C. Gong, W. Bao, G. Tang, Y. Jiang, and J. Liu, "A parallel algorithm for the two dimensional time fractional diffusion equation with implicit difference method," *The Scientific World Journal*, vol. 2014, Article ID 219580, 8 pages, 2014.

[33] C. Gong, W. Bao, G. Tang, Y. Jiang, and J. Liu, "A domain decomposition method for time fractional reaction-diffusion equation," *The Scientific World Journal*, vol. 2014, Article ID 681707, 5 pages, 2014.

[34] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.

[35] A. Fošner, "On the generalized Hyers-Ulam stability of module left (m, n)-derivations," *Aequationes Mathematicae*, vol. 84, no. 1-2, pp. 91–98, 2012.

[36] D. Popa, "Hyers-Ulam stability of the linear recurrence with constant coefficients," *Advances in Difference Equations*, no. 2, pp. 101–107, 2005.

[37] R. P. Agarwal, B. Xu, and W. Zhang, "Stability of functional equations in single variable," *Journal of Mathematical Analysis and Applications*, vol. 288, no. 2, pp. 852–869, 2003.