Egyptian Informatics Journal 24 (2023) 100408

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Two different parallel approaches for a hybrid fractional order Coronavirus model

N.H. Sweilam [a,*], S. Ahmed [a], Monika Heiner [b]

[a] Mathematics Department, Faculty of Science, Cairo University, Giza, Egypt
[b] Brandenburg Technical University (BTU), Cottbus, Germany

ARTICLE INFO

Keywords:
HPC
Parallel computing
MPI
GPU CUDA
Julia
Coronavirus mathematical model
Fractional predictor–corrector method

ABSTRACT

In this paper, two different parallel approaches for a hybrid fractional order Coronavirus (2019-nCov) mathematical model are presented. Both parallel approaches are implemented using Julia high level language. Parallel algorithm implementations are developed for the HPC cluster using Message Passing Interface (MPI) technology and general-purpose computing on GPUs (GPGPU) using Compute Unified Device Architecture (CUDA) based on hardware environments. The algorithm implementation are used to solve the real-world problem of the hybrid fractional order Coronavirus (2019-nCov) mathematical model and to study the parallel efficiency. The introduced hybrid fractional order derivative is defined as a linear combination of the integral of Riemann-Liouville and the fractional order Caputo derivative. A parallel algorithm is designed based on the predictor-corrector method with the discretization of the Caputo proportional constant fractional hybrid operator for solving the model problem numerically. Simulation results show that, both the new parallel approaches achieve significant efficiency.

1. Introduction

Fractional order derivatives are non-local in nature as the non-integer order operators in mathematical models help us comprehend things better. Models with fractional order derivatives are also better at capturing fading memory and crossover behaviour, as well as providing a higher accuracy level. It provides more information about predicting diseases under investigation ([1], [2], [3], [4], [5], [6], [19], [20], [21]). The operator of hybrid fractional can be defined as a linear combination of the Riemann-Liouville fractional integral and Caputo derivative. It is one of the most effective and dependable operators. It is more general than the Caputo fractional operator [13]. In addition, the hybrid fractional operator is a better fit for describing biological processes than the Caputo operator [15]. Moreover, optimal control theory has successful applications in biological and medical problems. Furthermore, fractional optimal control problems are a subclass of optimal control whose dynamics are described by fractional differential equations. We can minimize the impact of the COVID-19 pandemic by minimizing the number of detected asymptomatic infected people who acquire life-threatening symptoms and the threatened percentage of populations that become extinct. There are very interesting references in this field, for example, see ([15], [22–25]).

In literature, the growing demand for powerful computing technology has counted on increasing the number of cores rather than increasing the performance of a single unit. As a result, parallel computing has risen to prominence as a study topic capable of fulfilling time constraints. Both offline and simulations are possible. Despite the increasing coding difficulty due to the significant rise of computational scale, parallel programming has become increasingly popular [9].

The *de facto* C/C++/Fortran programming languages paired with MPI, OpenMP, pthread, Cilk, OpenCL, or CUDA are used as a standard for programming large-scale machines on distributed memory HPC Cluster, shared memory and heterogeneous hardware. Julia language is an interesting point on this spectrum due to it was designed by domain experts with scientific and technical computing. Julia is a high-level programming language with a focus on performance and productivity that was created for parallelism and scientific computing, making it a rival choice for developing HPC codes [17]. Since a lightweight API is provided by Julia to call C routines, Utilising C/C++ high-performance libraries has very little overhead and offers productive

* Corresponding author.
E-mail addresses: nsweilam@sci.cu.edu.eg (N.H. Sweilam), sameh@sci.cu.edu.eg (S. Ahmed), monika.heiner@b-tu.de (M. Heiner).

https://doi.org/10.1016/j.eij.2023.100408
Received 11 July 2023; Received in revised form 16 September 2023; Accepted 28 September 2023
Available online 12 October 2023

development in a high-level language that uses a lot less code to accomplish the similar results as C. [18]. At every parallelism level Julia provides built-in primitives for parallel computing by support features like dynamic types, meta-programming features, multiple dispatch, lightweight user threads, support for native code invocation, and packages for distributed, parallel computing.

Julia is Just-in-time (JIT) compiled language and uses LLVM compiler framework. Julia's JIT compiler converting high-level languages, on the fly at run time, into machine language that directly executed on the CPU for efficient subsequent execution by storing optimized code of a function was called. LLVM used as shared intermediate representation (IR) between different compiler optimization passes before being compiled to machine code. Julia executes native code and offers native ways to call functions of C or FORTRAN shared libraries. Additionally, it has native GPU support and offers a variety of libraries for using the GPU at various levels of abstraction.

Julia has set of libraries and packages that support parallel and distributed computing via Threads.jl, Distributed.jl and MPI.jl. Also, it has several packages for GPUs programming, CUDA.jl for NVIDIA GPUs, oneAPI.jl for Intel GPUs and AMDGPU.jl for AMD GPUs. Julia's JIT compiler integrate with these libraries to compile Julia code directly to CPU/GPU which make Julia powerful for programming on the GPU. Julia provides basic ahead-of-time compilation tools like PackageCompiler.jl which allows you to create a system image (serialized Julia sessions) to reduce needed compile times in every session and even enables you to create reproducible executable app used in a HPC cluster.

In work [8], the parallel computing algorithm was constructed based on linear tridiagonal equations for solving nonlinear time-space fractional partial differential equations. In [16], a parallel Crank–Nicolson finite-difference algorithm on a distributed system is presented for solving time-fractional parabolic equation. A parallel numerical algorithm for fractional-order systems of the Caputo-type derivatives was discussed in [10,11].

In this work, we construct parallel computing algorithm to solve the hybrid fractional order mathematical model of the Coronavirus (2019-nCov) based on predictor–corrector method (PCM). The discretization in this case of the Caputo proportional constant fractional hybrid operator (CPC-PCM) used for solving the model problem numerically. One of the advantages of the predictor-corrector method is the ease of its programming, and therefore it is considered one of the good methods for solving biological systems that are described in the form of nonlinear differential equations, but they usually take a large time to operate if we increase the number of points in order to obtain solutions that are accurate enough and to solve such a problem. In this paper, we presented one of the ways to improve the performance of this method. Julia high level language used to implement the parallel algorithm for the HPC cluster using Message Passing Interface (MPI) technology and general-purpose computing on GPUs (GPGPU) using Compute Unified Device Architecture (CUDA). The key objectives are:

- To Construct a parallel computing algorithm to solve the hybrid fractional order mathematical model of the Coronavirus (2019-nCov) using the predictor-corrector method.
- To implement the algorithm using Julia high-level language using Message Passing Interface (MPI) technology to enable parallel computing on an HPC cluster and Employ general-purpose computing on GPUs (GPGPU) using Compute Unified Device Architecture (CUDA) to enhance performance.
- Address the challenge of increased computational time when increasing the number of points for accurate solutions.

The structure of this paper is as follows. After introducing the issue in Section 1, Section 2 presents illustrations of the hybrid fractional-order derivative and the COVID-19 Model. In Section 3, a numerical approach to the CPC fractional-order problem is described, along with a thorough explanation, pseudo-code, and mechanism for implementing the parallel algorithms. A comparison of the designs and algorithm approaches described in Section 4 is offered. The Section 5 conclusions.

2. Notations & preliminaries

In this section, we will review several key definitions of fractional calculus that will be utilised throughout the rest of this article.

Definition 2.1. The left and right sides Riemann-Liouville derivatives of order α for a continuous function $f(t)$ defined as follows [7]

$$ {}_aD_t^{\alpha} f(t) = \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dt^n} \int_a^t \frac{f(s)}{(t-s)^{1-n+\alpha}} ds, \quad t > a, \tag{1} $$

$$ {}_tD_b^{\alpha} f(t) = \frac{1}{\Gamma(n-\alpha)} \frac{-d^n}{dt^n} \int_t^b \frac{f(s)}{(s-t)^{1-n+\alpha}} ds, \quad t < b. \tag{2} $$

where $-\infty < a < b < +\infty$, $\Omega = [a,b]$, $\alpha \in \mathbb{C}$, $\Re(\alpha) > 0$, $n = [\Re(\alpha)] + 1$.

Definition 2.2. The left and right sides of the Riemann-Liouville's integral of order α for a continuous function $f(t)$ are defined as follows [7]

$$ {}_aI_t^{\alpha} f(t) = \left[\int_a^t (t-s)^{\alpha-1} f(s) ds \right] \frac{1}{\Gamma(\alpha)}, \quad t > a, \tag{3} $$

$$ {}_tI_b^{\alpha} f(t) = \left[\int_t^b (t-s)^{\alpha-1} f(s) ds \right] \frac{1}{\Gamma(\alpha)}, \quad t < b. \tag{4} $$

where $-\infty < a < b < +\infty$, $\Omega = [a,b]$, $\alpha \in \mathbb{C}$, $\Re(\alpha) > 0$.

Definition 2.3. The left and right sides of Caputo's derivatives of order α for a function, $f(t)$, $f \in AC^n[a,b]$ [7] defined as follows

$$ ({}^C D_{a+}^{\alpha} f)(t) = ({}_a^C D_t^{\alpha} f)(t) = \frac{1}{\Gamma(n-\alpha)} \int_a^t \frac{f^{(n)}(s)}{(t-\xi)^{1-n+\alpha}} ds, \quad t > a, \tag{5} $$

$$ ({}^C D_{b-}^{\alpha} f)(x) = ({}_t^C D_b^{\alpha} f)(t) = \frac{(-1)^n}{\Gamma(n-\alpha)} \int_t^b \frac{f^(n)(s)}{(s-t)^{1-n+\alpha}} ds, \quad t < b. \tag{6} $$

where $-\infty < a < b < +\infty$, $\Omega = [a,b]$, $\alpha \in \mathbb{C}$, $n = [\Re(\alpha)] + 1$, $\Re(\alpha) \notin \mathbb{N}_0$.

Definition 2.4. [13] We define the hybrid Caputo proportional fractional operator (CP) as follows:

$$ \begin{aligned} {}_0^{CP} D_t^{\alpha} y(t) &= \left(\int_0^t (-s+t)^{-\alpha} (y'(s) K_0(s,\alpha) + y(s) K_1(s,\alpha)) ds \right) \frac{1}{\Gamma(1-\alpha)}, \\ &= (K_1(t,\alpha) y(t) + K_0(t,\alpha) y'(t)) \left(\frac{t^{-\alpha}}{\Gamma(1-\alpha)} \right), \end{aligned} \tag{7} $$

where, $K_0(t,\alpha) = \alpha t^{(1-\alpha)}$, $K_1(t,\alpha) = \frac{(1-\alpha)}{t^{-\alpha}}$ $0 < \alpha < 1$.

Alternatively, the hybrid Caputo proportional constant fractional operator (CPC) can be used [13]:

$$ \begin{aligned} {}_0^{CPC} D_t^{\alpha} y(t) &= (\Gamma(1-\alpha))^{-1} \left(\int_0^t (t-s)^{-\alpha(t)} (y'(s) K_0(\alpha) + K_1(\alpha) y(s)) ds \right) \\ &= K_0(\alpha) {}_0^C D_t^{\alpha} y(t) + K_1(\alpha) {}_0^{RL} I_t^{1-\alpha} y(t), \end{aligned} \tag{8} $$

and, $(1-\alpha) Q^{\alpha} = K_1(\alpha)$, $\alpha Q^{(1-\alpha)} = K_0(\alpha)$, Q is a constant.

Table 1
System (9) variables [12].

variables	Description
F	The category of fatality.
H	The hospitalized category.
E	The exposed category.
I	The category of infectious and category.
R	The category of recovery.
S	The category of susceptible.
A	The category of infectious but asymptomatic.
P	The super-spreaders category.

2.1. COVID-19 model

In this part, we apply a newly proposed hybrid fractional order derivative to the Coronavirus spreading model from [12]. Eight nonlinear differential equations make up this model. The dimension of the left side would be $(time)^{-\alpha}$, if we switched the equations' order to $alpha$. We should adjust the parameters' dimensions to fit the dimensions. Additionally, the fractional order system is converted to a classical one when $\alpha \to 1$. Let us assume that $\delta_i^\alpha = \delta_p^\alpha = \delta_h^\alpha = 0$. All variables and parameter descriptions are provided in Tables 1 and 2. After that, the modified model is given as follows:

$$
\begin{aligned}
{}_0^{CPC}D_t^\alpha S &= -\beta^\alpha \frac{IS}{N} - l\beta^\alpha \frac{HS}{N} - \beta'^\alpha \frac{PS}{N},\\
{}_0^{CPC}D_t^\alpha E &= \beta^\alpha \frac{IS}{N} + l\beta^\alpha \frac{HS}{N} + \beta'^\alpha \frac{PS}{N} - K^\alpha E,\\
{}_0^{CPC}D_t^\alpha I &= K^\alpha \rho_1 E - (\gamma_a^\alpha + \gamma_i^\alpha) I - \delta_i^\alpha I,\\
{}_0^{CPC}D_t^\alpha P &= K^\alpha \rho_2 E - (\gamma_a^\alpha + \gamma_i^\alpha) P - \delta_p^\alpha P,\\
{}_0^{CPC}D_t^\alpha A &= K^\alpha (1-\rho_1-\rho_2) E,\\
{}_0^{CPC}D_t^\alpha H &= \gamma_a^\alpha (I+P) - \gamma_r^\alpha H - \delta_h^\alpha H,\\
{}_0^{CPC}D_t^\alpha R &= \gamma_i^\alpha (I+P) + \gamma_r^\alpha H,\\
{}_0^{CPC}D_t^\alpha F &= \delta_i^\alpha I + \delta_p^\alpha P + \delta_h^\alpha H.
\end{aligned} \tag{9}
$$

3. Parallel algorithm of hybrid fractional order

In this section, first, we use the predictor-corrector method to numerically solve the hybrid fractional order derivatives of COVID-19 Model (Equation (9)). Second, we show the description of our proposed parallel algorithm.

3.1. Numerical method for the CPC fractional-order equation

In this subsection, the predictor-corrector Scheme is used to study numerically the hybrid fractional order derivatives of COVID-19 Model (Equation (9)).

Consider the general form of fractional differential equation with CPC operator is as follows:

$$
{}_0^{CPC}D_t^\alpha y(t) = \xi(t, y(t)), \quad 0 < \alpha \le 1, \quad y(0) = y_0. \tag{10}
$$

By using (8) and GL-approximation to approximate Caputo's fractional derivatives:

$$
\begin{aligned}
&\frac{Q^\alpha(-\alpha+1)}{\tau^{\alpha-1}\Gamma(2-\alpha)} \sum_{i=0}^{1+n} y_{n-i+1} \left[(1+i)^{(-\alpha+1)} - (i)^{(-\alpha+1)} \right]\\
&+ \frac{\alpha Q^{(-\alpha+1)}}{\tau^\alpha} \left(y_{n+1} - \sum_{i=1}^{n+1} \mu_i(\alpha) y_{n+1-i} - q_{n+1} y_0 \right) = \xi(t_i, y(t_i)),
\end{aligned} \tag{11}
$$

and, $\frac{\alpha}{Q^{(\alpha-1)}} = K_0(\alpha)$, $K_1(\alpha) = \frac{(1-\alpha)}{Q^{-\alpha}}$, and we can put

$$
r_1 = \frac{Q^\alpha(1-\alpha)}{\tau^{\alpha-1}\Gamma(2-\alpha)}, \qquad r_2 = \frac{Q^{(1-\alpha)}}{\tau^\alpha}.
$$

Also, $\tau = \frac{T_f}{N_n}$, $N_n \in \mathbb{N}$. $\mu_i(\alpha) = (-1)^{i-1} \binom{\alpha}{i}$, $\mu_1 = \alpha$, $q_i = (\Gamma(1-\alpha))^{-1} i^\alpha$ $i = 1, 2, ..., n+1$. Also, ([14]):

$$
\begin{aligned}
&0 < \mu_{i+1} < \mu_i < ... < \mu_1 = \alpha < 1,\\
&0 < q_{i+1} < q_i < ... < q_1 = \frac{1}{\Gamma(1-\alpha)}.
\end{aligned}
$$

We can rewrite (11) as Predictor–corrector method as follows:

$$
y_{1+n}^* = \frac{\left\{ \begin{array}{l} \xi(t_n, y_n) - r_1 \sum_{i=1}^{n+1} y_{n-i+1} \left[(1+i)^{(-\alpha+1)} - (i)^{(-\alpha+1)} \right] \\ + r_2 \sum_{i=1}^{n+1} q_{n+1} y_0 + \mu_i y_{n+1-i} \end{array} \right\}}{r_1 + r_2}. \tag{12}
$$

Predictor

$$
y_{n+1} = \frac{\left\{ \begin{array}{l} \xi(t_{n+1}, y_{n+1}^*) - r_1 \sum_{i=1}^{n+1} y_{n-i+1} \left[(1+i)^{(-\alpha+1)} - (i)^{(-\alpha+1)} \right] \\ + r_2 \sum_{i=1}^{n+1} \mu_i(\alpha) y_{n+1-i} + q_{n+1} y_0 \end{array} \right\}}{r_1 + r_2}. \tag{13}
$$

(Corrector)

Now, we can put

$$
\left[(1+i)^{(-\alpha+1)} - (i)^{(-\alpha+1)} \right] = P_i(\alpha),
$$

$$
I_{RM} = r_1 \sum_{i=1}^{n+1} y_{n-i+1} P_i(\alpha), \qquad D_{CAP} = r_2 \sum_{i=1}^{n+1} y_0 q_{n+1} + \mu_i(\alpha) y_{n+1-i} + .
$$

Then we can write the hybrid Scheme with Predictor–corrector as follows:

$$
y_{n+1}^* = \frac{\xi(t_n, y_n) - I_{RM} + D_{CAP}}{r_1 + r_2}. \tag{14}
$$

Table 2
The Coronavirus model's parameter values [12].

Parameter	Description	Values (per $day^{-\alpha}$)
β^α	The transmission coefficient from diseased persons	$(255/100)^\alpha$
l	Hospitalised individuals' relative transmissibility	156/100 dimensionless
β'^α	Transmission coefficient as a result of super-spreaders	$(765/100)^\alpha$
K^α	The rate at which those who have been exposed become infectious	$(25/100)^\alpha$
ρ_1	The rate at which people who have been exposed become infected	580/1000 dimensionless
ρ_2	The rate at which individuals who have been exposed become super-spreaders.	1/1000 dimensionless
γ_a^α	Hospitalisation rates	$(94/100)^\alpha$
γ_i^α	Recovery rate without hospitalization	$(27/100)^\alpha$
γ_r^α	Hospitalised patients' recovery rate	$(5/10)^\alpha$
δ_i^α	Disease-related mortality rate due to contaminated class	$(35/10)^\alpha$
δ_p^α	The death rate from disease caused by super-spreaders	1^α
δ_h^α	Disease-related death rate in the hospitalised class	$(3/10)^\alpha$

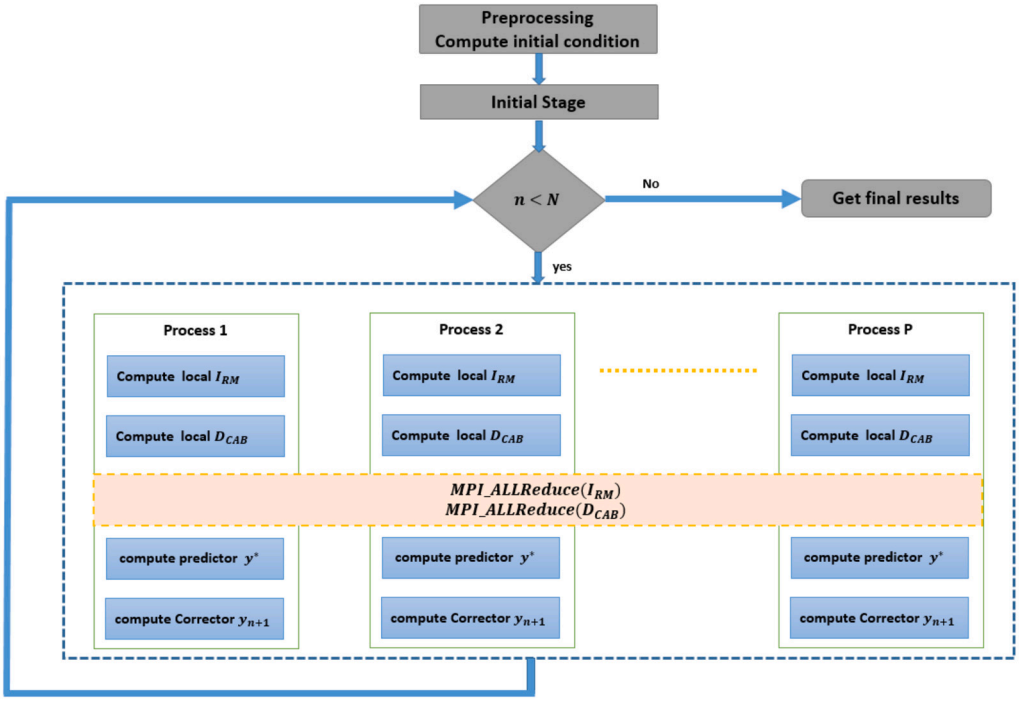

Fig. 1. General flow chart of the parallel algorithm for hybrid fractional order.

Predictor

$$y_{n+1} = \frac{\xi(t_{n+1}, y^*_{n+1}) - I_{RM} + D_{CAP}}{r_1 + r_2}. \tag{15}$$

(Corrector)

Remark 1. We can claim the Caputo's discrete with Predictor–corrector method if we put in (11), $K_0(\alpha) = 1$, $K_1(\alpha) = 0$.

3.2. Parallel algorithm for hybrid fractional order

In this subsection, the description of the proposed parallel hybrid fractional order (PHFO) algorithm is presented in Algorithm 1, and a general flow chart diagram is presented in Fig. 1.

$P(\alpha)$, $\mu(\alpha)$ and q are computed in the initializing stage lines 3-6 to reduce computation time. At each time step n problem was decomposed according to number of available processes p to sub tasks lines 9-10. Each available process will compute local I_{RM} and local D_{CAP} lines 13-16. In order to obtain load balancing, all processes will act as master node so local I_{RM} and local D_{CAP} will reduced to all available process by using $MPI.ALLreduce()$ function and each process will compute global I_{RM} and global D_{CAP} lines 17-18 which will be used to update system local at each process using predictor-corrector method line 19-20. In case n less than the number of processes p, the rest of processes will be idle.

The graphics processing unit (GPU) is massively parallel processors. GPU is comprised of a set of Streaming Multi-Processors (SM), which form a grid of threads organized into blocks. Several concurrent thread blocks can be running in each SM. To fully utilise all of these threads, multiple thread blocks must be used to run the CUDA programme code under the single instruction multiple data (SIMD) paradigm. The adaptive CUDA approach code consists of following steps:

1. Transferring y_0, $P(\alpha)$, $\mu(\alpha)$ and q data to the global GPU memory.
2. For every time step iteration running the CUDA kernel core.
   - Compute local I_{RM} and D_{CAP}
   - Reducing I_{RM} and D_{CAP}
   - Update next y_n on GPU memory
3. Transferring the final results from the GPU to the CPU memory.

Algorithm 1 Parallel algorithm for hybrid fractional order.

1: Input data: end of time T, Number of points N, Number of process p and Current process c_P
2: $y_0 \leftarrow$ initial condition of system
3: **for** $i = 1 : N$ **do** ▷ Calculate $P_i(\alpha), \mu_i(\alpha) and q_i$
4: $P_i(\alpha) \leftarrow (1+i)^{(-\alpha+1)} - (i)^{(-\alpha+1)}$
5: $\mu_i(\alpha) \leftarrow (-1)^{i-1}\binom{\alpha}{i}$
6: $q_i \leftarrow \frac{i^\alpha}{\Gamma(1-\alpha)}$
7: **end for**
8: **for** Every time step $n < N$ **do**
9: $n_{start} \leftarrow \lfloor n/p \rfloor c_p$
10: $n_{end} \leftarrow \lfloor n/p \rfloor (c_p + 1)$
11: $I_{RM} \leftarrow 0$
12: $D_{CAP} \leftarrow 0$
13: **for** $k = n_{start} : n_{end}$ **do**
14: $I_{RM} \leftarrow I_{RM} + y_{k-i+1} P_k(\alpha)$ ▷ compute local I_{RM} at c_P
15: $D_{CAP} \leftarrow D_{CAP} + \mu_k(\alpha) + y_{k-i+1} - q_{k+1} y_0$ ▷ compute local D_{CAP} at c_P
16: **end for**
17: $I_{RM} \leftarrow MPI.ALLreduce(I_{RM})$ ▷ compute global I_{RM} and broadcast to all process
18: $D_{CAP} \leftarrow MPI.ALLreduce(D_{CAP})$ ▷ compute global D_{CAP} and broadcast to all process
19: $y^* \leftarrow f(t_n, y_n) - r_1 I_{RM} + r_2 D_{CAP}/(r_1 + r_2)$ ▷ compute the predictor
20: $y_{n+1} \leftarrow f(t_{n+1}, y^*_{n+1}) - r_1 I_{RM} + r_2 D_{CAP}/(r_1 + r_2)$ ▷ compute the corrector
21: **end for**

4. Computational and experimental results

This section presents the implementation results for the parallelization method for solving hybrid fractional order derivatives of COVID-19 Model (9) at $\alpha = 0.9$ using the parameters in Table 2. The initial conditions given as follows $E(0) = 0$, $R(0) = 0$, $S(0) = N - 6$, $F(0) = 0$, $P(0) = 5$, $I(0) = 1$, $H(0) = 0$, $A(0) = 0$. The parallel execution times Tp of the proposed algorithm implementation with different N problem sizes using different p numbers of processes are presented. The parallelization speedup (Sp) and the efficiency of parallelization (E) are used to examine the varying effectiveness of parallel execution time. The speedup

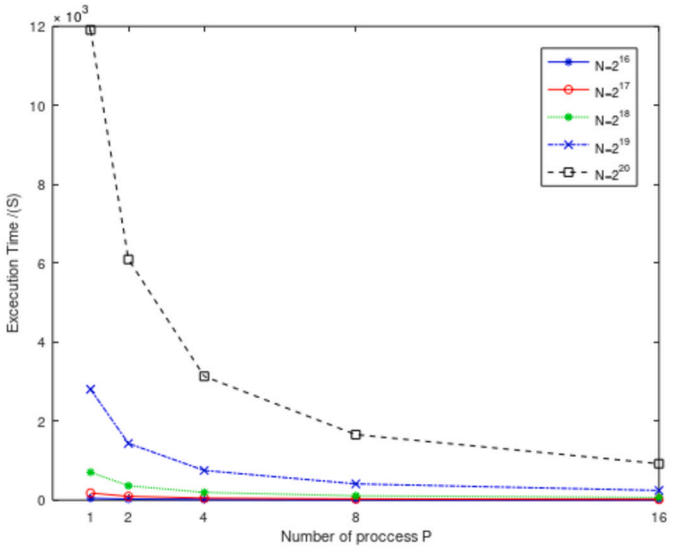

Fig. 2. Execution time for PHFO algorithm.

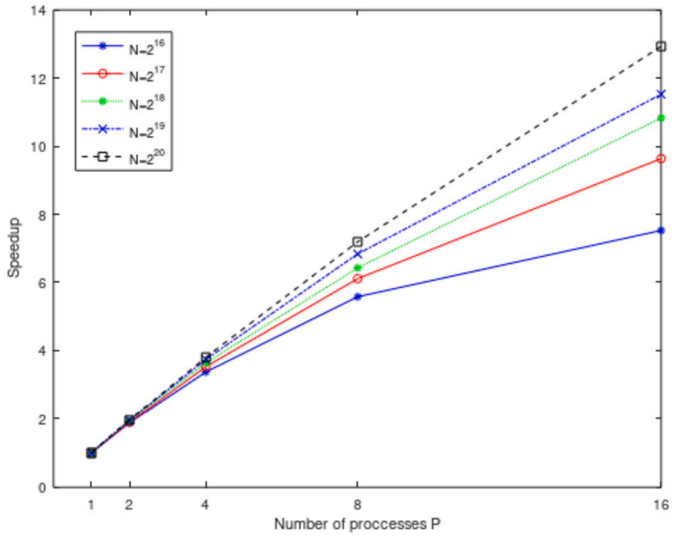

Fig. 3. Speedup for PHFO algorithm.

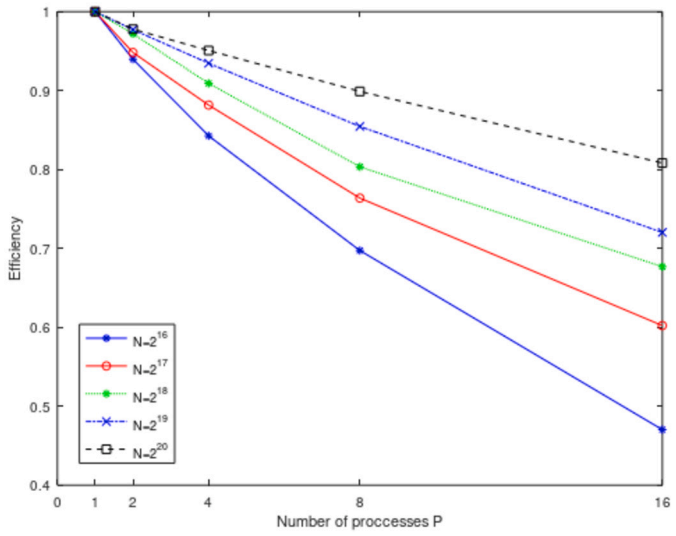

Fig. 4. Efficiency for PHFO algorithm.

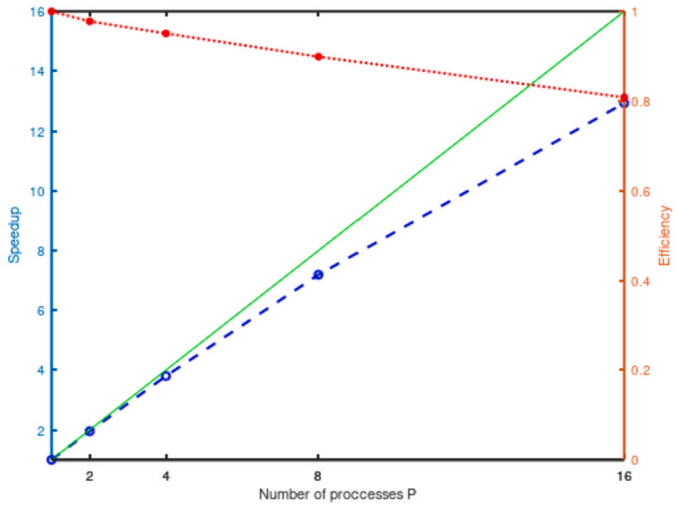

Fig. 5. The speedup (blue dashed) and the efficiency (red dots) of the parallelization depending on the number of processes (P) for the PHFO algorithm at $N = 2^{20}$; the maximum possible speedup is indicated by a green solid line.

(Sp) is determined as the ratio of the time it takes for a programme to run on one node (T1) to the execution time on P processes (Tp). The efficiency of the parallelization is calculated as the speedup (Sp) divided by the number of processes (P).

Julia has two popular distribution modes: MPI.jl and Distributed.jl. Message Passing Interface (MPI) is a standard Julia wrapper built on the single-program multiple-data (SPMD) paradigm. It is the most ideal choice for large-scale distributed parallel computing. It is very flexible and easy to use it from Julia and get similar C performance. Distributed.jl is the method to spawn new processes in separate memory spaces. Supports distributed array operations for distributed memory machines, its master-slave distributed architecture makes it good for many-core and multi-processor systems. Julia supports native GPU, we will use CUDA.jl the basic programming interface essential to using NVIDIA CUDA GPUs in Julia. CUDA.jl provides a lower-level approach for writing kernels for fine-grained control and user-friendly high-level abstractions.

The MPI- and CUDA-based environments have been used for computational analysis. Proposed algorithm simulation was implemented and compiled using Julia programming language version 1.7.2 using MPI.jl and CUDA.jl packages. The serial performance results were obtained on a single core.

4.1. MPI-based environments

MPI-based environments simulations were run on the HPC system provided by faculty of nanotechnology Cairo university Cluster. Each node has two sockets, each with an Intel(R) Xeon(R) Gold 6130 @ 2.10 GHz with 96 GiB of memory.

Fig. 2 present the parallel execution time of run time results (in seconds) using various time step sizes (number of global points). As we can see in Figs. 2 execution times decreases with increasing number of processes.

Fig. 3 presents experimental speedup curves for various time step sizes using increasing number of processes. As we can see, that the effectiveness of the parallelization increases with increasing considered length. Fig. 4 show resulting an efficiency about 80% using 16 processes.

The results in Figs. 2, 3, 4 and 5 demonstrate the Execution time, speedup and efficiency of our parallel MPI implementation for solving hybrid fractional-order systems in Julia. Our implementation provides a significant speedup over a serial implementation and maintains high efficiency values for a range of numbers of processes. This makes it a

Table 3
GPU time in seconds for the solution of the model (9).

Number of steps (N)	CUDA running time	CUDA last step running time
2^{16}	4.739	0.000105
2^{17}	11.048	0.000122
2^{18}	43.109	0.000324
2^{19}	171.283	0.000523
2^{20}	710.877	0.001061

valuable tool for solving fractional-order systems in parallel and can be particularly useful for large systems or systems that require high computational resources.

4.2. CUDA-based environments

CUDA-based environments simulations was run on the HPC system provided by SRTA CITY Cluster Gpu machine PowerEdge Dell 740XD (2× Intel Xeon Gold 6248) with Tesla V100 GPU (5120 CUDA cores, 32 GB RAM).

Table 3 present the CUDA execution time results (in seconds) and execution time for last time step iterations at a different number of problem size N = 2^{16}, 2^{17}, 2^{18}, 2^{19} and 2^{20} (Fig. 6). At N= 2^{20} CUDA implementation obtains speedup S ≈ 16. Fig. 7 presents comparing execution time for CUDA approach and MPI with 16 processes approach implementations.

4.3. Numerical results and analysis

We used CPC-PCM to solve the hybrid fractional-order system (9) in order to execute the numerical simulations in the following section. We use the values of parameters in Table 2, with the initial conditions $E(0) = 0,\ R(0) = 0,\ S(0) = N - 6,\ F(0) = 0,\ P(0) = 5,\ I(0) = 1,\ H(0) = 0,\ A(0) = 0$. Wuhan has a population of approximately 11 million people. The individual movement was restricted in the city during the COVID-19 outbreak because of quarantine. As a result, a restriction on the disease spread was provided. We examine the whole population in our model, and $N = \frac{11000000}{250}$ [12] and $0 < \alpha \leq 1$ and $0 < Q \leq 1$. Fig. 8, illustrates the numerical answers for the suggested model's (9) behaviour at different values of α and $\delta_I = \delta_p = \delta_h = 0$. This figure describes how solutions' behaviour alters when we use different values α. The fractional model gave us more details about spreading the disease. Moreover, this model is more general than the classical model of integer order.

5. Conclusions

Two parallel approaches for solving the mathematical model of the hybrid fractional order Coronavirus (2019-nCov) are constructed. The parallel approaches are implemented on a distributed system using a message-passing interface (MPI) and A GPU-CUDA Framework. The advantages and comparison of the parallel approaches over the sequential are discussed. CPC derivative is defined as a linear combi-

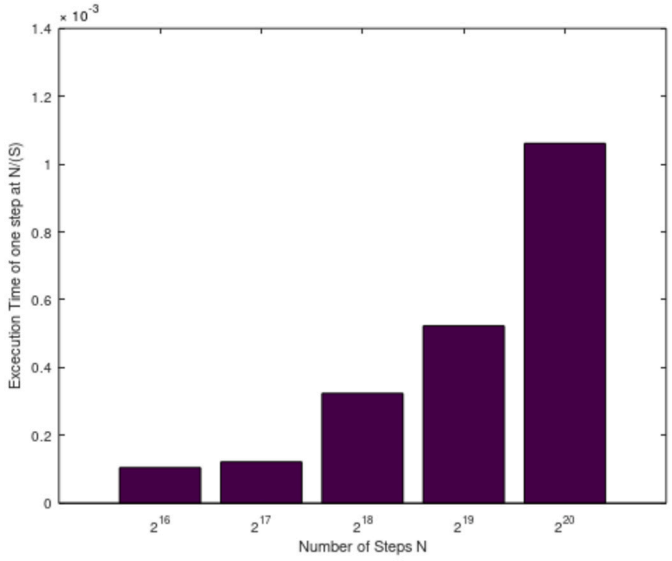

Fig. 6. Execution times of last step for different problem sizes N.

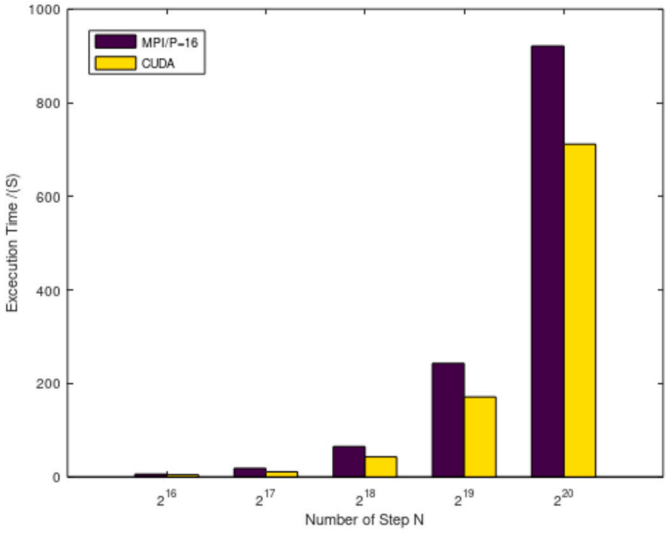

Fig. 7. Execution times of CUDA and MPI 16 processes for different problem sizes N.

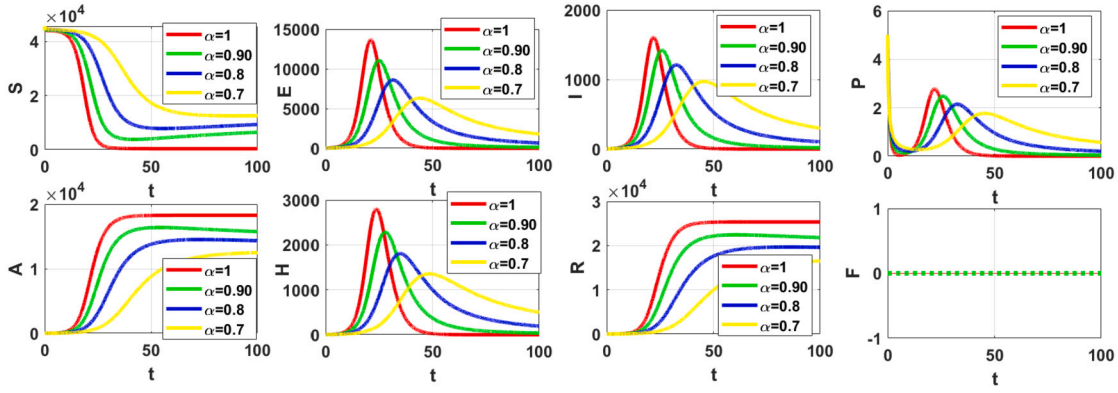

Fig. 8. The behaviour numerical solutions for the model (9).

nation of the integral of Riemann-Liouville and the fractional order Caputo derivative. Two parallel approaches are constructed based on the predictor-corrector method with the Caputo proportional constant fractional hybrid operator's discretization for solving the model's problem numerically. Numerical experiments using a different number of nodes of the Intel(R) Xeon(R) Gold 6130 Linux PC Cluster and Tesla V100 GPU (5120 CUDA cores, 32 GB RAM) were used to test the efficiency of the developed parallel algorithms, which delivered a considerable performance boost in terms of execution times.

CRediT authorship contribution statement

N.H. Sweilam: Methodology, Supervision, Validation, Writing – review & editing. **S. Ahmed:** Conceptualization, Data curation, Methodology, Software, Writing – original draft. **Monika Heiner:** Investigation, Visualization, Writing – review & editing.

Declaration of competing interest

There is no competing interest.

References

[1] Agarwal P, Ramadan MA, Rageh AAM, Hadhoud AR. A fractional-order mathematical model for analyzing the pandemic trend of COVID-19. Math Methods Appl Sci 2021. https://doi.org/10.1002/mma.8057.

[2] Atangana A, Araz SI. A novel Covid-19 model with fractional differential operators with singular and non-singular kernels: analysis and numerical scheme based on Newton polynomial. Alex Eng J 2021;60(4):3781–806. https://doi.org/10.1016/j.aej.2021.02.016.

[3] Pinto CMA, Tenreiro Machado JA, Burgos-Simón C. Modified SIQR model for the COVID-19 outbreak in several countries. Math Methods Appl Sci 2022. https://doi.org/10.1002/mma.8082.

[4] Yadav S, Kumar D, Singh J, Baleanu D. Analysis and dynamics of fractional order Covid-19 model with memory effect. Results Phys 2021;24:104017.

[5] Zhang L, Rahman MU, Ahmad S, Riaz MB, Jarad F. Dynamics of fractional order delay model of coronavirus disease. AIMS Math 2021;7(3):4211–32.

[6] Sweilam NH, AL Mekhlafi SM, Almutairi A, Baleanu D. A hybrid fractional COVID-19 model with general population mask use: numerical treatments. Alex Eng J 2021;60(30):1–14. https://doi.org/10.1016/j.aej.2021.01.057.

[7] Podlubny I. Fractional differential equations. New York: Academic Press; 1999.

[8] Biala TA, Khaliq AQM. Parallel algorithms for nonlinear time-space fractional parabolic PDEs. J Comput Phys 2018;375:135–54.

[9] Pacheco P. An introduction to parallel programming. Burlington, NJ, USA: Morgan Kaufmann; 2011.

[10] Bonchis C, Kaslik E, Rosu F. HPC optimal parallel communication algorithm for the simulation of fractional-order systems. J Supercomput 2019;75:1014–25.

[11] Wang Q, Liu J, Gong C, Tang X, Fu G, Xing Z. An efficient parallel algorithm for Caputo fractional reaction-diffusion equation with implicit finite-difference method. Adv Differ Equ 2016;207.

[12] Ndärou F, Area I, Nieto JJ, Torres DFM. Mathematical modeling of COVID-19 transmission dynamics with a case study of Wuhan. Chaos Solitons Fractals 2020. https://doi.org/10.1016/j.chaos.2020.109846.

[13] Baleanu D, Fernandez A, Akgül A. On a fractional operator combining proportional and classical differintegrals. Mathematics 2020;8(3). https://doi.org/10.3390/math8030360.

[14] Scherer R, Kalla S, Tang Y, Huang J. The Grünwald-Letnikov method for fractional differential equations. Comput Math Appl 2011;62:902–17.

[15] Sweilam NH, Al-Mekhlafi SM. A survey on numerical studies for fractional biological models and their optimal control. In: Radwan Ahmad G, Khanday Farooq A, Said Lobna A, editors. Fractional order systems; an overview of mathematics, design, and applications for engineers. Academic Press. ISBN 978-0-12-824293-3, 2022. Chapter 1.

[16] Sweilam NH, Moharram H, Moniem NKA, Ahmed S. A parallel Crank–Nicolson finite difference method for time-fractional parabolic equation. J Numer Math 2014;22(4):363–82. De Gruyter.

[17] Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: a fresh approach to numerical computing. SIAM Rev 2017;59(1):65–98.

[18] Hunold Sascha, Steiner Sebastian. Benchmarking Julia's communication performance: is Julia HPC ready or full HPC. In: 2020 IEEE/ACM performance modeling, benchmarking and simulation of high performance computer systems (PMBS); 2020. p. 20–5.

[19] Kumar P, Rangaig NA, Abboubakar H, Kumar A, Manickam A. Prediction studies of the epidemic peak of coronavirus disease in Japan: from Caputo derivatives to Atangana-Baleanu derivatives. Int J Model Simul Sci Comput 2022;13(1):2250012.

[20] Abboubakar H, Fandio R, Sofack BS, Ekobena Fouda HP. Fractional dynamics of measles epidemic model. Axioms 2022;11(8):363. https://doi.org/10.3390/axioms11080363.

[21] Kumar P, Erturk VS, Govindaraj V, Inc M, Abboubakar H, Nisar KS. Dynamics of COVID-19 epidemic via two different fractional derivatives. Int J Model Simul Sci Comput 2023;14(3):2350007.

[22] Marzban HR. Optimal control of nonlinear fractional order delay systems governed by Fredholm integral equations based on a new fractional derivative operator. ISA Trans 2023;133:233–47.

[23] Marzban HR, Nezami A. Analysis of nonlinear fractional optimal control systems described by delay Volterra-Fredholm integral equations via a new spectral collocation method. Chaos Solitons Fractals 2022;162:112499.

[24] Sabermahani S, Ordokhani Y, Rahimkhani P. Application of generalized Lucas wavelet method for solving nonlinear fractal-fractional optimal control problems. Chaos Solitons Fractals 2023;170:113348.

[25] Dehestani H, Ordokhani Y. An efficient approach based on Legendre-Gauss-Lobatto quadrature and discrete shifted Hahn polynomials for solving Caputo-Fabrizio fractional Volterra partial integro-differential equations. J Comput Appl Math 2022;403:11385.