Egyptian Informatics Journal 24 (2023) 100408

Contents lists available at [ScienceDirect](http://www.ScienceDirect.com/)

Egyptian Informatics Journal

journal homepage: [www.sciencedirect.com](http://www.sciencedirect.com/)

Two different parallel approaches for a hybrid fractional order Coronavirus  model

N.H. Sweilam [a](#_bookmark0)*,*[∗](#_bookmark3), S. Ahmed [a](#_bookmark0), Monika Heiner [b](#_bookmark1)

a *Mathematics Department, Faculty of Science, Cairo University, Giza, Egypt*

b *Brandenburg Technical University (BTU), Cottbus, Germany*

A R T I C L E I N F O A B S T R A C T

*Keywords:*

HPC

Parallel computing MPI

GPU CUDA

Julia

Coronavirus mathematical model Fractional predictor–corrector method

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.

# 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 bet- ter 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](#_bookmark31)], [[2](#_bookmark33)], [[3](#_bookmark35)], [[4](#_bookmark38)], [[5](#_bookmark39)], [[6](#_bookmark40)], [[19](#_bookmark32)], [[20](#_bookmark34)], [[21](#_bookmark36)]). 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](#_bookmark22)]. In addition, the hybrid fractional operator is a better fit for describing biological processes than the Caputo operator [[15](#_bookmark23)]. Moreover, optimal control theory has successful applications in biological and medical problems. Furthermore, fractional optimal control problems are a subclass of op- timal 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 refer- ences in this field, for example, see ([[15](#_bookmark23)], [[22](#_bookmark37)–[25](#_bookmark41)]).

In literature, the growing demand for powerful computing tech- nology has counted on increasing the number of cores rather than increasing the performance of a single unit. As a result, parallel comput- ing has risen to prominence as a study topic capable of fulfilling time constraints. Both oﬄine and simulations are possible. Despite the in- creasing coding diﬃculty due to the significant rise of computational scale, parallel programming has become increasingly popular [[9](#_bookmark43)].

The *de facto* C/C++/Fortran programming languages paired with MPI, OpenMP, pthread, Cilk, OpenCL, or CUDA are used as a stan- dard 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 productiv- ity that was created for parallelism and scientific computing, making it a rival choice for developing HPC codes [[17](#_bookmark24)]. 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](mailto:nsweilam@sci.cu.edu.eg) (N.H. Sweilam), [sameh@sci.cu.edu.eg](mailto:sameh@sci.cu.edu.eg) (S. Ahmed), [monika.heiner@b-tu.de](mailto: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

1110-8665/© 2023 THE AUTHORS. Published by Elsevier BV on behalf of Faculty of Computers and Artificial Intelligence, Cairo University. This is an open access article under the CC BY-NC-ND license ([http://creativecommons.org/licenses/by-nc-nd/4.0/).](http://creativecommons.org/licenses/by-nc-nd/4.0/))

development in a high-level language that uses a lot less code to accom- plish the similar results as C. [[18](#_bookmark30)]. At every parallelism level Julia pro- vides built-in primitives for parallel computing by support features like dynamic types, meta-programming features, multiple dispatch, light- weight user threads, support for native code invocation, and packages for distributed, parallel computing.

Julia is Just-in-time (JIT) compiled language and uses LLVM com- piler 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 represen-

tation (IR) between different compiler optimization passes before being

ing the parallel algorithms. A comparison of the designs and algorithm approaches described in Section [4](#_bookmark13) is offered. The Section [5](#_bookmark19) conclusions.

# Notations & preliminaries

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

of order *𝛼* for a continuous function *𝑓* (*𝑡*) defined as follows [[7](#_bookmark42)] **Definition 2.1.** The left and right sides Riemann-Liouville derivatives

*𝑡*

*𝛼* 1

*𝑑𝑛*

*𝑓* (*𝑠*)

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 dis- tributed computing via Threads.jl, Distributed.jl and MPI.jl. Also, it has

*𝑎𝐷𝑡 𝑓* (*𝑡*)= Γ(*𝑛* − *𝛼*) *𝑑𝑡𝑛* ∫

*𝑎*

*𝑏*

*𝑡𝐷𝑏 𝑓* (*𝑡*)= Γ(*𝑛* − *𝛼*) *𝑑𝑡𝑛* ∫

*𝛼* 1 −*𝑑𝑛*

*𝑡*

(*𝑡* − *𝑠*)1−*𝑛*+*𝛼 𝑑𝑠, 𝑡 > 𝑎,* (1)

*𝑓* (*𝑠*)

(*𝑠* − *𝑡*)1−*𝑛*+*𝛼 𝑑𝑠, 𝑡 < 𝑏.* (2)

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 PackageCom- piler.jl which allows you to create a system image (serialized Julia

where −∞ *< 𝑎< 𝑏<* +∞, Ω = [*𝑎, 𝑏*], *𝛼* ∈ ℂ*,* ℜ(*𝛼*) *>* 0, *𝑛* = [ℜ(*𝛼*)] + 1.

tegral of order *𝛼* for a continuous function *𝑓* (*𝑡*) are defined as follows **Definition 2.2.** The left and right sides of the Riemann-Liouville’s in-

[[7](#_bookmark42)]

sessions) to reduce needed compile times in every session and even en- ables you to create reproducible executable app used in a HPC cluster.

In work [[8](#_bookmark44)], the parallel computing algorithm was constructed based on linear tridiagonal equations for solving nonlinear time-space frac-

tional partial differential equations. In [[16](#_bookmark25)], a parallel Crank–Nicolson

*𝑡*

*𝑎𝐼𝑡 𝑓* (*𝑡*)= ∫ (*𝑡* − *𝑠*)

[

*𝛼 𝛼*−1

*𝑎*

*𝑏*

[

*𝛼 𝛼*−1

*𝑓* (*𝑠*)*𝑑𝑠*] 1 *, 𝑡 > 𝑎,* (3)

] 1

Γ(*𝛼*)

finite-difference algorithm on a distributed system is presented for solv- ing time-fractional parabolic equation. A parallel numerical algorithm for fractional-order systems of the Caputo-type derivatives was dis-

*𝑡𝐼𝑏 𝑓* (*𝑡*)=

∫ (*𝑡* − *𝑠*)

*𝑡*

*𝑓* (*𝑠*)*𝑑𝑠*

Γ(*𝛼*) *, 𝑡 < 𝑏.* (4)

cussed in [[10](#_bookmark26),[11](#_bookmark27)].

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 oper-

where −∞ *< 𝑎< 𝑏<* +∞, Ω = [*𝑎, 𝑏*], *𝛼* ∈ ℂ*,* ℜ(*𝛼*) *>* 0.

**Definition 2.3.** The left and right sides of Caputo’s derivatives of order

*𝛼* for a function, *𝑓* (*𝑡*), *𝑓* ∈ *𝐴𝐶𝑛*[*𝑎, 𝑏*] [[7](#_bookmark42)] defined as follows

*𝑡*

ator (CPC-PCM) used for solving the model problem numerically. One

(*𝐶 𝛼*

*𝐶 𝛼*

1 *𝑓* (*𝑛*)(*𝑠*)

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 nonlin- ear differential equations, but they usually take a large time to operate

*𝐷𝑎*+*𝑓* )(*𝑡*)= (*𝑎 𝐷𝑡 𝑓* )(*𝑡*)= Γ(*𝑛* − *𝛼*) ∫

*𝑎*

*𝑏*

*𝑛*

(*𝐶 𝛼 𝐶 𝛼*  (−1)

*𝐷 𝑓* )(*𝑥*)= ( *𝐷 𝑓* )(*𝑡*)=

(*𝑡* − *𝜉*)1−*𝑛*+*𝛼 𝑑𝑠, 𝑡 > 𝑎,* (5)

*𝑓* (*𝑛 𝑠*

)( )

*𝑑𝑠, 𝑡 < 𝑏.* (6)

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

*𝑏*−

*𝑡 𝑏*

Γ(*𝑛* − *𝛼*) ∫

*𝑡*

(*𝑠* − *𝑡*)1−*𝑛*+*𝛼*

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:

(

where −∞ *< 𝑎< 𝑏<* +∞, Ω = [*𝑎, 𝑏*], *𝛼* ∈ ℂ, *𝑛* = [ℜ(*𝛼*)] + 1, ℜ(*𝛼*) ∉ ℕ0.

**Definition 2.4.** [[13](#_bookmark22)] We define the hybrid Caputo proportional frac- tional operator (CP) as follows:

*𝑡*

0 *𝑡* ∫

1

Γ(1 − *𝛼*)

)

* To Construct a parallel computing algorithm to solve the hybrid

fractional order mathematical model of the Coronavirus (2019-

*𝐶𝑃 𝐷𝛼𝑦*(*𝑡*)= (−*𝑠* + *𝑡*)−*𝛼* (*𝑦*′(*𝑠*)*𝐾*0(*𝑠, 𝛼*)+ *𝑦*(*𝑠*)*𝐾*1(*𝑠, 𝛼*))*𝑑𝑠 ,*

0

nCov) using the predictor-corrector method.

=( (

) ( )+ (

) ′( ))( *𝑡*−*𝛼* )

* To implement the algorithm using Julia high-level language us- ing Message Passing Interface (MPI) technology to enable parallel

*𝐾*1 *𝑡, 𝛼 𝑦 𝑡*

*𝐾*0 *𝑡, 𝛼 𝑦 𝑡*

Γ(1 − *𝛼*)

*,* (7)

computing on an HPC cluster and Employ general-purpose comput- ing on GPUs (GPGPU) using Compute Unified Device Architecture (CUDA) to enhance performance.

* Address the challenge of increased computational time when in- creasing the number of points for accurate solutions.

where, *𝐾*0(*𝑡, 𝛼*) = *𝛼𝑡*(1−*𝛼*), *𝐾*1(*𝑡, 𝛼*) = (1−*𝛼*) 0 *< 𝛼 <* 1.

Alternatively, the hybrid Caputo proportional constant fractional op- erator (CPC) can be used [[13](#_bookmark22)]:

*𝑡*−*𝛼*

*𝑡*

(

)

0 *𝑡* ∫

The structure of this paper is as follows. After introducing the issue in Section [1](#_bookmark2), Section [2](#_bookmark4) presents illustrations of the hybrid fractional- order derivative and the COVID-19 Model. In Section [3](#_bookmark9), a numerical

*𝐶𝑃 𝐶 𝐷𝛼𝑦*(*𝑡*) = (Γ(1 − *𝛼*))−1 (*𝑡* − *𝑠*)−*𝛼*(*𝑡*)(*𝑦*′(*𝑠*)*𝐾*0(*𝛼*)+ *𝐾*1(*𝛼*)*𝑦*(*𝑠*))*𝑑𝑠*

0

= *𝐾*0(*𝛼*)*𝐶 𝐷𝛼𝑦*(*𝑡*)+ *𝐾*1(*𝛼*)*𝑅𝐿𝐼* 1−*𝛼𝑦*(*𝑡*)*,* (8)

approach to the CPC fractional-order problem is described, along with

0 *𝑡*

0 *𝑡*

a thorough explanation, pseudo-code, and mechanism for implement-

and, (1 − *𝛼*)*𝑄𝛼* = *𝐾*1(*𝛼*), *𝛼𝑄*(1−*𝛼*) = *𝐾*0(*𝛼*), *𝑄* is a constant.

**Table 1**

System ([9](#_bookmark8)) variables [[12](#_bookmark28)].

variables Description

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

*𝐶𝑃 𝐶 𝐷𝛼𝑦*(*𝑡*)= *𝜉*(*𝑡, 𝑦*(*𝑡*))*,* 0 *< 𝛼* ≤ 1*, 𝑦*(0) = *𝑦 .* (10)

*𝐹* The category of fatality.

*𝐻* The hospitalized category.

*𝐸* The exposed category.

0 *𝑡* 0

By using ([8](#_bookmark5)) and GL-approximation to approximate Caputo’s fractional derivatives:

*𝐼* The category of infectious and category.

*𝑅* The category of recovery.

*𝑆* The category of susceptible.

*𝐴* The category of infectious but asymptomatic.

*𝛼* 1+*𝑛*

*𝜏𝛼*−1Γ(2 − *𝛼*) *𝑖*=0

*𝑄* (−*𝛼* + 1) ∑

(

*𝑦𝑛*−*𝑖*+1

[(1 + *𝑖*)(−*𝛼*+1)

– (*𝑖*)(−*𝛼*+1)]

*𝑃* The super-spreaders category.

+ *𝛼𝑄*(−*𝛼*+1)

*𝜏𝛼*

*𝑦𝑛*+1 −

*𝑛*+1

*𝑖*=1

∑

*𝜇𝑖*(*𝛼*)*𝑦𝑛*+1−*𝑖* − *𝑞𝑛*+1*𝑦*0)

= *𝜉*(*𝑡𝑖, 𝑦*(*𝑡𝑖*))*,* (11)

* 1. *COVID-19 model*

and,

*𝛼*

*𝑄*(*𝛼*−1)

= *𝐾*0(*𝛼*), *𝐾*1(*𝛼*) = (1−*𝛼*) , and we can put

In this part, we apply a newly proposed hybrid fractional order derivative to the Coronavirus spreading model from [[12](#_bookmark28)]. Eight non- linear differential equations make up this model. The dimension of the

*𝑄𝛼*(1 − *𝛼*)

*𝑟*1 *𝜏𝛼*−1Γ(2 − *𝛼*) *, 𝑟*2

= =

*𝑇𝑓*

*𝑄*(1−*𝛼*)

*𝜏𝛼 .*

*𝑄*−*𝛼*

*𝑖*−1( *𝛼* )

−1 *𝛼*

left side would be (*𝑡𝑖𝑚𝑒*)−*𝛼* , if we switched the equations’ order to *𝑎𝑙𝑝ℎ𝑎*.

We should adjust the parameters’ dimensions to fit the dimensions. Ad-

ditionally, the fractional order system is converted to a classical one

when *𝛼* → 1. Let us assume that *𝛿𝛼* = *𝛿𝛼* = *𝛿𝛼* = 0. All variables and pa-

Also, *𝜏* = , *𝑁𝑛* ∈ ℕ. *𝜇𝑖*(*𝛼*) = (−1)

*𝑛*

*𝑁*

*𝑖* = 1*,* 2*, ..., 𝑛* + 1. Also, ([[14](#_bookmark29)]):

0 *< 𝜇𝑖*+1 *< 𝜇𝑖 < ... < 𝜇*1 = *𝛼<* 1*,*

*𝑖* , *𝜇*1 = *𝛼*, *𝑞𝑖* = (Γ(1 − *𝛼*)) *𝑖*

*𝑖 𝑝 ℎ*

rameter descriptions are provided in Tables [1](#_bookmark6) and [2](#_bookmark10). After that, the modified model is given as follows:

*𝐶𝑃 𝐶 𝐷𝛼𝑆* =− *𝛽𝛼 𝐼𝑆* − *𝑙𝛽𝛼 𝐻𝑆* − *𝛽*′ *𝛼 𝑃 𝑆 ,*

0 *< 𝑞 < 𝑞 < ... < 𝑞* = 1 *.*

Γ(1 − *𝛼*)

*𝑖*+1 *𝑖* 1

We can rewrite ([11](#_bookmark7)) as Predictor–corrector method as follows:

0 *𝑡*

*𝑁*

*𝐼𝑆*

*𝑁*

*𝐻𝑆*

*𝑁*

*𝑃 𝑆*

⎧ ( )− *𝑟*

∑*𝑛*+1 *𝑦*

[(1 + *𝑖*)(−*𝛼*+1) − (*𝑖*)(−*𝛼*+1)] ⎫

*𝐶𝑃 𝐶 𝐷𝛼𝐸* =*𝛽𝛼*

+ *𝑙𝛽𝛼*

+ *𝛽*′ *𝛼*

– *𝐾𝛼𝐸,*

⎨⎪ *𝜉 𝑡𝑛, 𝑦𝑛*

1 *𝑖*=1

*𝑛*−*𝑖*+1 ⎪⎬

0 *𝑡 𝑁*

*𝐶𝑃 𝐶 𝛼 𝛼*

0 *𝐷𝑡 𝐼* =*𝐾 𝜌*1*𝐸* − (*𝛾𝑎* + *𝛾𝑖* )*𝐼* − *𝛿𝑖 𝐼,*

*𝐶𝑃 𝐶 𝐷𝛼𝑃* =*𝐾𝛼𝜌*2*𝐸* − (*𝛾𝛼* + *𝛾𝛼*)*𝑃* − *𝛿𝛼𝑃,*

*𝑁 𝑁*

*𝛼 𝛼 𝛼*

⎪⎩ +*𝑟*2 ∑*𝑛*+1 *𝑞𝑛*+1*𝑦*0 + *𝜇𝑖𝑦𝑛*+1−*𝑖* ⎪⎭

0 *𝑡*

*𝑦*

∗ 1+*𝑛*

= *𝑖*=1 *.* (12)

*𝑟*1 + *𝑟*2

*𝑎 𝑖 𝑝*

Predictor

*𝐶𝑃 𝐶 𝐷𝛼𝐴* =*𝐾𝛼*(1 − *𝜌*1 − *𝜌*2)*𝐸,*

⎧ [ ] ⎫

0 *𝑡*

*𝐶𝑃 𝐶 𝛼 𝛼*

0 *𝐷𝑡 𝐻* =*𝛾𝑎* (*𝐼* + *𝑃* )− *𝛾𝑟 𝐻* − *𝛿ℎ𝐻,*

⎪

*𝛼 𝛼*

⎪ *𝜉 𝑡𝑛*+1*, 𝑦*

)− *𝑟*1 ∑*𝑛*+1 *𝑦𝑛*−*𝑖*+1

(1 + *𝑖*)(−*𝛼*+1) − (*𝑖*)(−*𝛼*+1)

*𝐶𝑃 𝐶 𝐷𝛼𝑅* =*𝛾𝛼*(*𝐼* + *𝑃* )+ *𝛾𝛼𝐻,*

( ∗

*𝑛*+1

*𝑖*=1

0

*𝑡*

*𝑖*

*𝑟*

*𝑦*

= *𝑖*=1 *.* (13)

*𝑟*1 + *𝑟*2

⎨⎪⎩ +*𝑟*2 ∑*𝑛*+1 *𝜇𝑖*(*𝛼*)*𝑦𝑛*+1−*𝑖* + *𝑞𝑛*+1*𝑦*0

⎬⎪⎭

0 *𝑡 𝑖 𝑝 ℎ*

*𝑛*+1

*𝐶𝑃 𝐶 𝐷𝛼𝐹* =*𝛿𝛼𝐼* + *𝛿𝛼𝑃* + *𝛿𝛼𝐻.* (9)

(Corrector)

# Parallel algorithm of hybrid fractional order

In this section, first, we use the predictor-corrector method to nu- merically solve the hybrid fractional order derivatives of COVID-19 Model (Equation ([9](#_bookmark8))). Second, we show the description of our proposed parallel algorithm.

*𝑖*

Now, we can put

[(1 + *𝑖*)(−*𝛼*+1) − (*𝑖*)(−*𝛼*+1)] = *𝑃𝑖*(*𝛼*)*,*

*𝑛*+1

∑

*𝐼* = *𝑟 𝑦 𝑃* (*𝛼*)*, 𝐷*

*𝑅𝑀*

1

*𝑖*=1

*𝑛*−*𝑖*+1

*𝑖*

*𝐶𝐴𝑃*

*𝑛*+1

= *𝑟 𝑦 𝑞*

2

*𝑖*=1

0 *𝑛*+1

∑

+ *𝜇* (*𝛼*)*𝑦* + *.*

* 1. *Numerical method for the CPC fractional-order equation*

*𝑛*+1−*𝑖*

In this subsection, the predictor-corrector Scheme is used to study

Then we can write the hybrid Scheme with Predictor–corrector as fol- lows:

numerically the hybrid fractional order derivatives of COVID-19 Model (Equation ([9](#_bookmark8))).

∗

*𝑛*+1

*𝑦*

= *𝜉*(*𝑡𝑛, 𝑦𝑛*)− *𝐼𝑅𝑀* + *𝐷𝐶𝐴𝑃 .* (14)

*𝑟*1 + *𝑟*2

**Table 2**

The Coronavirus model’s parameter values [[12](#_bookmark28)].

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Values (per *𝑑𝑎𝑦*−*𝛼* ) |
| *𝛽𝛼* | The transmission coeﬃcient from diseased persons | (255∕100)*𝛼* |
| *𝑙* | Hospitalised individuals’ relative transmissibility | 156∕100 dimensionless |
| *𝛽*′ *𝛼* | Transmission coeﬃcient as a result of super-spreaders | (765∕100)*𝛼* |
| *𝐾𝛼* | The rate at which those who have been exposed become infectious | (25∕100)*𝛼* |
| *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 |
| *𝛾𝛼*  *𝛾𝛼*  *𝛾𝛼*  *𝛿𝛼*  *𝛿𝛼* | Hospitalisation rates  Recovery rate without hospitalization Hospitalised patients’ recovery rate  Disease-related mortality rate due to contaminated class  The death rate from disease caused by super-spreaders | (94∕100)*𝛼*  (27∕100)*𝛼* (5∕10)*𝛼* (35∕10)*𝛼* 1*𝛼* |
| *𝛿𝛼* | Disease-related death rate in the hospitalised class | (3∕10)*𝛼* |

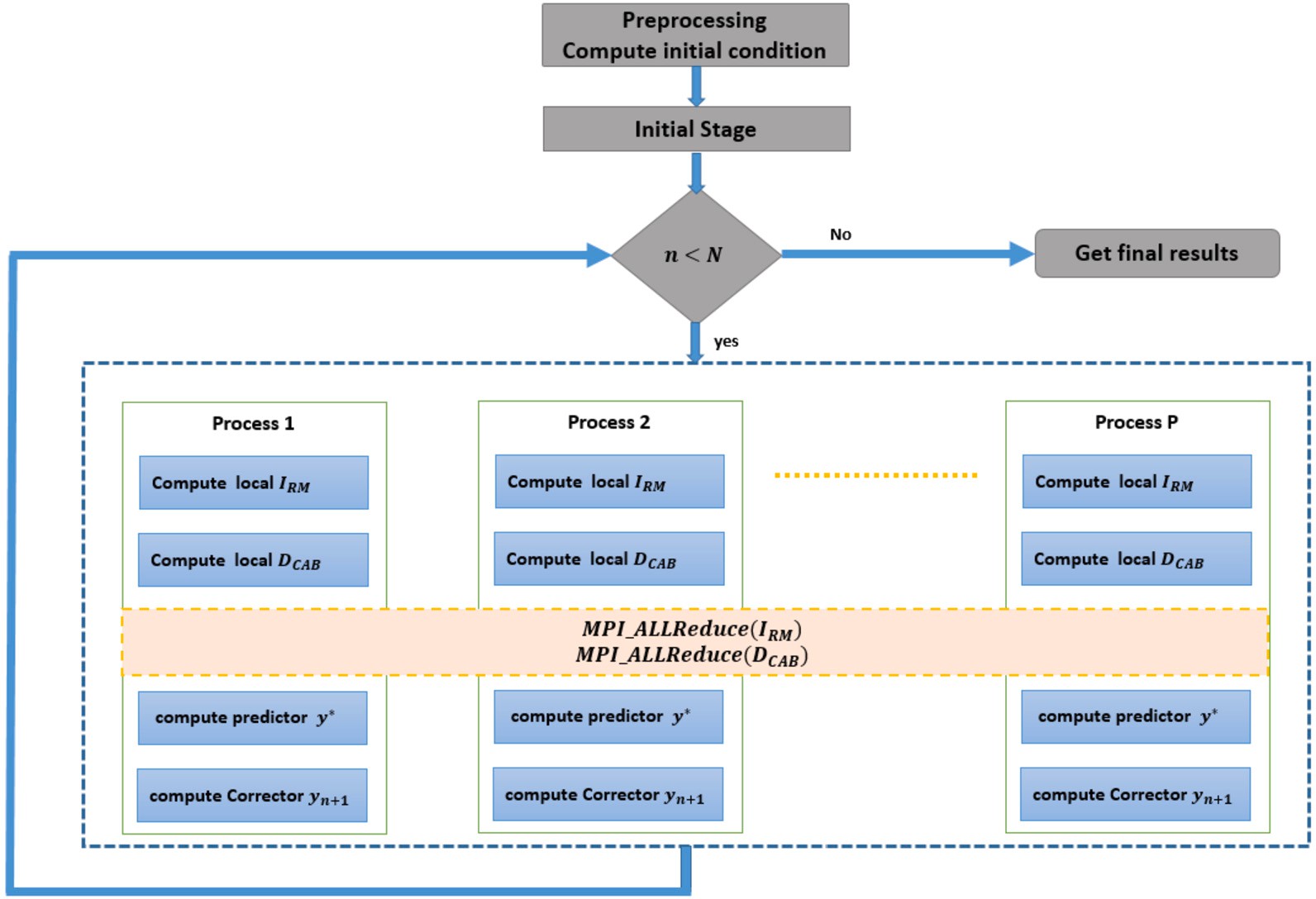
*𝑎*

*𝑖*

*𝑟*

*𝑖*

*𝑝 ℎ*



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

Predictor

*𝜉*(*𝑡𝑛*+1*, 𝑦*∗

)− *𝐼𝑅𝑀* + *𝐷𝐶𝐴𝑃*

**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

*𝑦𝑛*+1

= *𝑛*+1 *.* (15)

*𝑟*1 + *𝑟*2

process *𝑐𝑃*

2: *𝑦*0 ← initial condition of system

(Corrector)

method if we put in ([11](#_bookmark7)), *𝐾*0(*𝛼*) = 1, *𝐾*1(*𝛼*) = 0. **Remark 1.** We can claim the Caputo’s discrete with Predictor–corrector

* 1. *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](#_bookmark11), and a general flow chart diagram is presented in Fig. [1](#_bookmark12).

*𝑃* (*𝛼*), *𝜇*(*𝛼*) 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 *𝐼𝑅𝑀* and local *𝐷𝐶𝐴𝑃* lines

13-16. In order to obtain load balancing, all processes will act as master

node so local *𝐼* and local *𝐷* will reduced to all available process

3: **for** *𝑖* = 1 : *𝑁* **do** *⊳* Calculate *𝑃𝑖* (*𝛼*)*, 𝜇𝑖* (*𝛼*)*𝑎𝑛𝑑𝑞𝑖*

4: *𝑃𝑖* (*𝛼*) ← (1 + *𝑖*)(−*𝛼*+1) − (*𝑖*)(−*𝛼*+1)

5: *𝜇𝑖* (*𝛼*) ← (−1) ( )

*𝑖*−1 *𝛼*

*𝑖*

6: *𝑞* ← *𝑖𝛼*

Γ(1−*𝛼*)

*𝑖*

7: **end for**

8: **for** Every time step *𝑛* < *𝑁* **do**

9: *𝑛𝑠𝑡𝑎𝑟𝑡* ← *𝑛*∕*𝑝 𝑐𝑝*

⌊ ⌋

10: *𝑛𝑒𝑛𝑑* ← *𝑛 𝑝*

⌊

∕*𝑝* (*𝑐* + 1)

⌋

11: *𝐼𝑅𝑀* ← 0

12: *𝐷𝐶𝐴𝑃* ← 0

13: **for** *𝑘* = *𝑛𝑠𝑡𝑎𝑟𝑡* ∶ *𝑛𝑒𝑛𝑑* **do**

14: *𝐼𝑅𝑀* ← *𝐼𝑅𝑀* + *𝑦𝑘*−*𝑖*+1 *𝑃𝑘* (*𝛼*) *⊳* compute local *𝐼𝑅𝑀* at *𝑐𝑃*

15: *𝐷𝐶𝐴𝑃* ← *𝐷𝐶𝐴𝑃* + *𝜇𝑘* (*𝛼*) + *𝑦𝑘*−*𝑖*+1 − *𝑞𝑘*+1 *𝑦*0 *⊳* compute local *𝐷𝐶𝐴𝑃* at *𝑐𝑃*

16: **end for**

17: *𝐼𝑅𝑀* ← *𝑀𝑃 𝐼 .𝐴𝐿𝐿𝑟𝑒𝑑𝑢𝑐𝑒*(*𝐼𝑅𝑀* ) *⊳* compute global *𝐼𝑅𝑀* and broadcast to all

process

18: *𝐷𝐶𝐴𝑃* ← *𝑀𝑃 𝐼 .𝐴𝐿𝐿𝑟𝑒𝑑𝑢𝑐𝑒*(*𝐷𝐶𝐴𝑃* ) *⊳* compute global *𝐷𝐶𝐴𝑃* and broadcast to all

process

19: *𝑦*∗ ← *𝑓* (*𝑡𝑛 , 𝑦𝑛* ) − *𝑟*1 *𝐼𝑅𝑀* + *𝑟*2 *𝐷𝐶𝐴𝑃* ∕(*𝑟*1 + *𝑟*2 ) *⊳* compute the predictor

*𝑅𝑀*

*𝐶𝐴𝑃*

20: *𝑦𝑛*+1 ← *𝑓* (*𝑡𝑛*+1 *, 𝑦*∗

) − *𝑟*1 *𝐼𝑅𝑀* + *𝑟*2 *𝐷𝐶𝐴𝑃* ∕(*𝑟*1 + *𝑟*2 ) *⊳* compute the corrector

by using *𝑀𝑃 𝐼 .𝐴𝐿𝐿𝑟𝑒𝑑𝑢𝑐𝑒*() function and each process will compute global *𝐼𝑅𝑀* and global *𝐷𝐶𝐴𝑃* lines 17-18 which will be used to update

system local at each process using predictor-corrector method line 19-

21: **end for**

*𝑛*+1

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 adap- tive CUDA approach code consists of following steps:

1. Transferring *𝑦*0, *𝑃* (*𝛼*), *𝜇*(*𝛼*) and *𝑞* data to the global GPU memory.
2. For every time step iteration running the CUDA kernel core.
   * Compute local *𝐼𝑅𝑀* and *𝐷𝐶𝐴𝑃*
   * Reducing *𝐼𝑅𝑀* and *𝐷𝐶𝐴𝑃*
   * Update next *𝑦𝑛* on GPU memory
3. Transferring the final results from the GPU to the CPU memory.

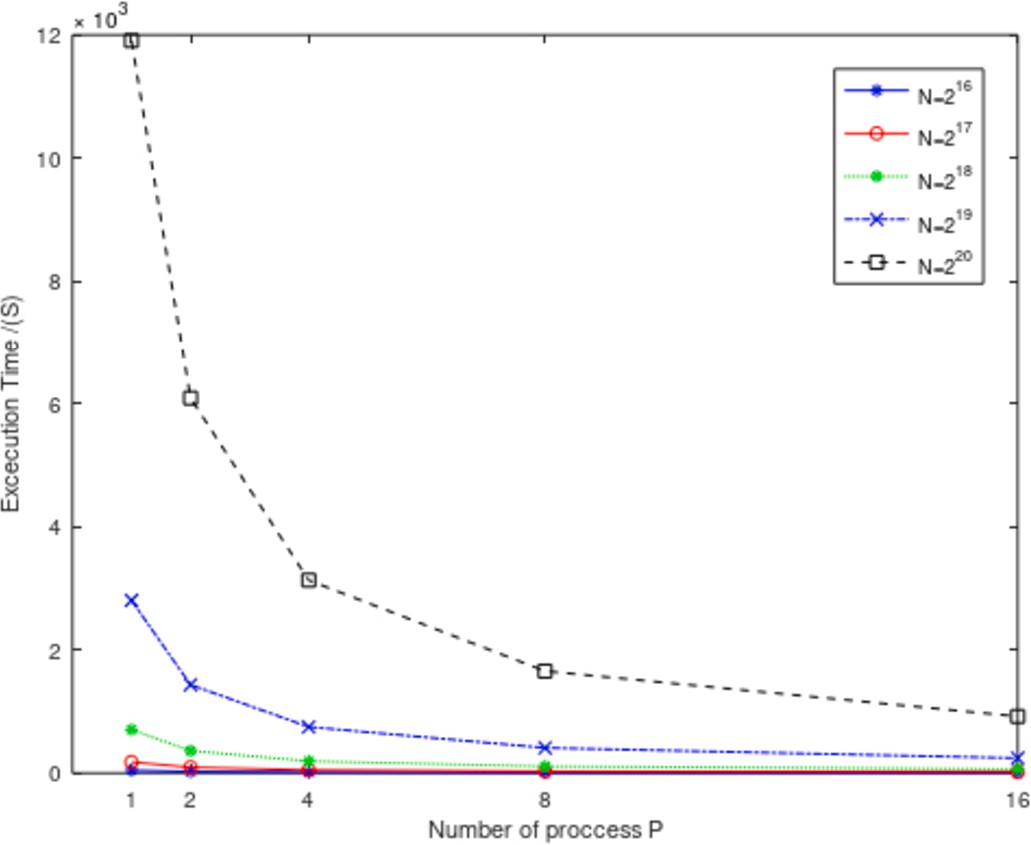
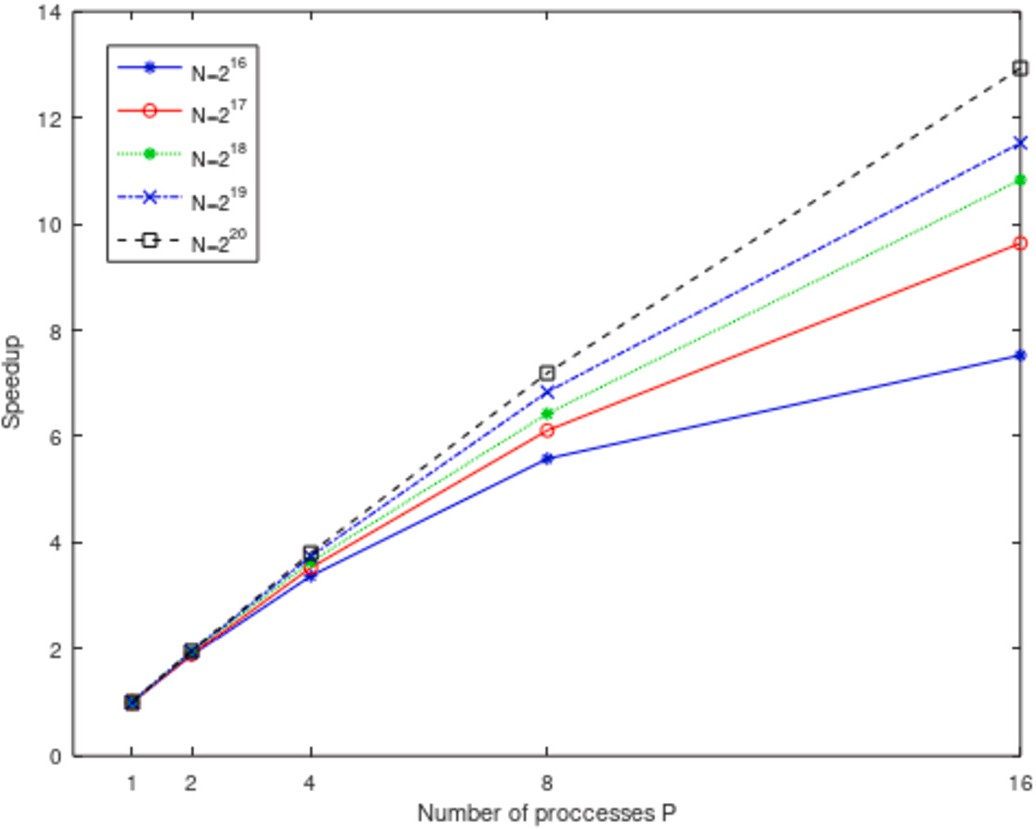
# Computational and experimental results

This section presents the implementation results for the paralleliza-

Model ([9](#_bookmark8)) at *𝛼* = 0*.*9 using the parameters in Table [2](#_bookmark10). The initial condi- tion method for solving hybrid fractional order derivatives of COVID-19 tions 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 us- ing different *𝑝* numbers of processes are presented. The parallelization

speedup (Sp) and the eﬃciency of parallelization (E) are used to exam- ine the varying effectiveness of parallel execution time. The speedup

**Fig. 2.** Execution time for PHFO algorithm.

(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 ef- ficiency 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 flexi- ble and easy to use it from Julia and get similar C performance. Dis- tributed.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 compu- tational 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.

* 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](#_bookmark14) 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](#_bookmark14) execution times decreases with increasing number of processes.

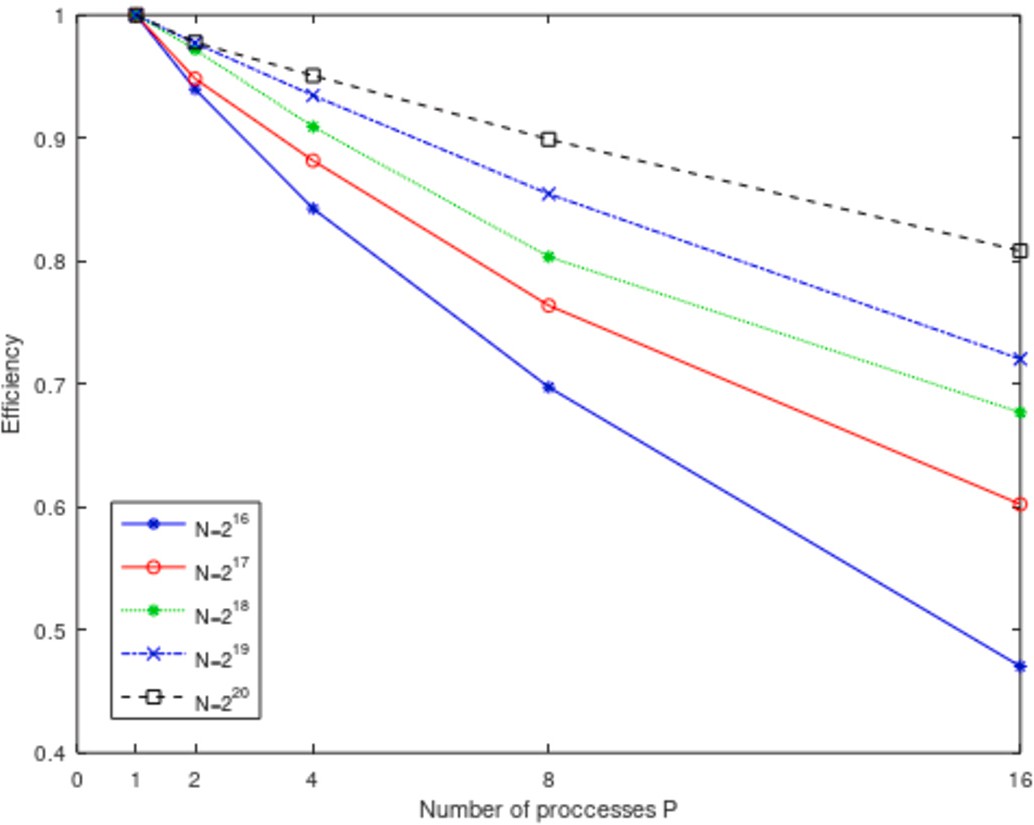
Fig. [3](#_bookmark15) presents experimental speedup curves for various time step sizes using increasing number of processes. As we can see, that the ef-

length. Fig. [4](#_bookmark16) show resulting an eﬃciency about 80% using 16 pro- fectiveness of the parallelization increases with increasing considered

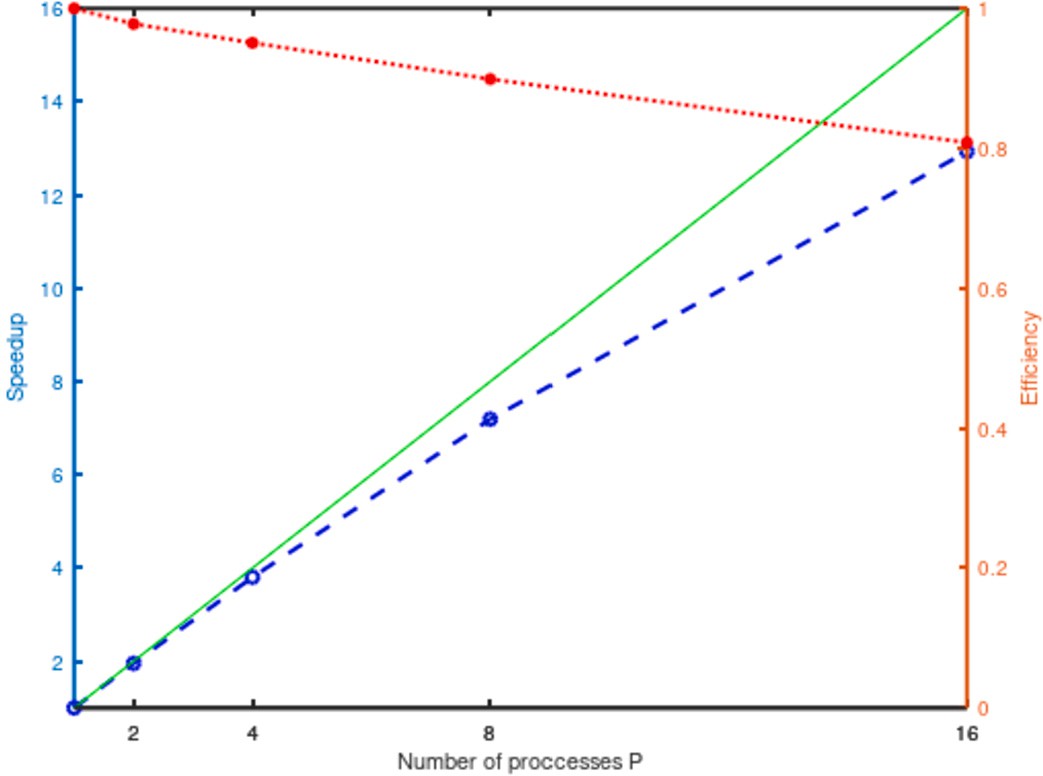
cesses.

The results in Figs. [2](#_bookmark14), [3](#_bookmark15), [4](#_bookmark16) and [5](#_bookmark17) 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

**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 paral- lelization depending on the number of processes (P) for the PHFO algorithm at

N =220 ; the maximum possible speedup is indicated by a green solid line.

**Table 3**

GPU time in seconds for the solution of the model ([9](#_bookmark8)).

Number of steps (N) CUDA running time CUDA last step running time

|  |  |  |
| --- | --- | --- |
| 216 | 4*.*739 | 0*.*000105 |
| 217 | 11*.*048 | 0*.*000122 |
| 218 | 43*.*109 | 0*.*000324 |
| 219 | 171*.*283 | 0*.*000523 |
| 220 | 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.

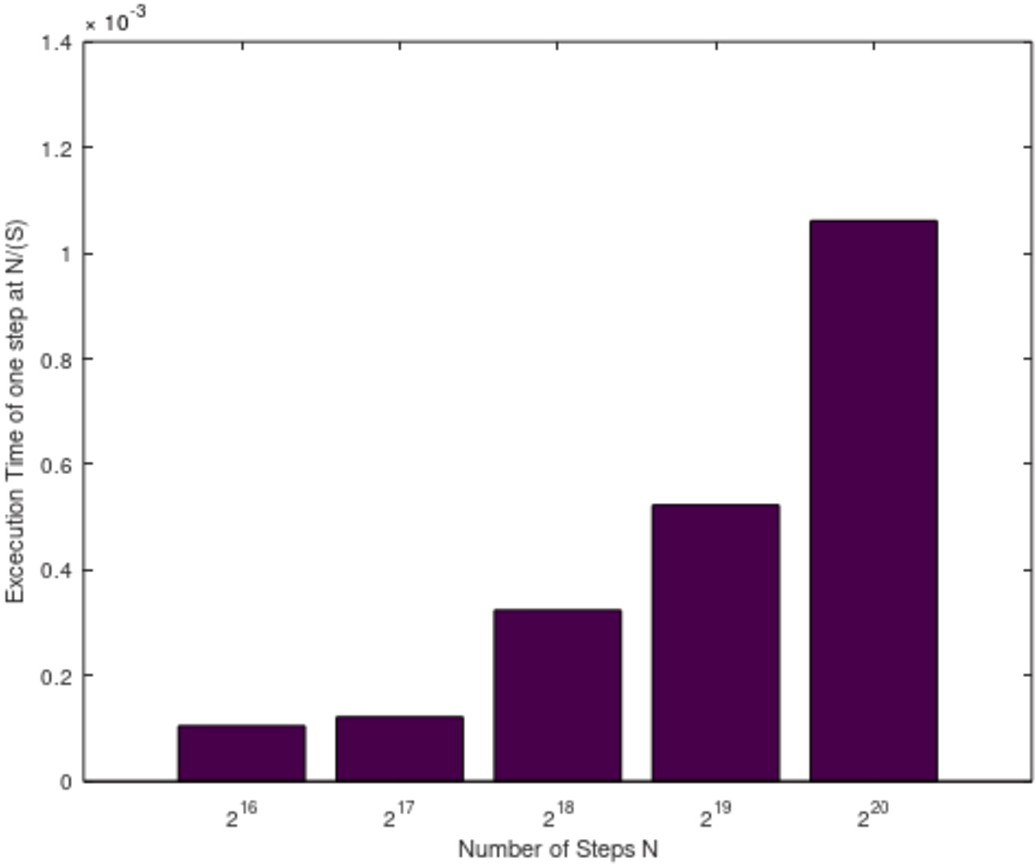
* 1. *CUDA-based environments*

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](#_bookmark18) present the CUDA execution time results (in seconds) and

problem size N = 216 , 217 , 218 , 219 and 220 (Fig. [6](#_bookmark20)). At N= 220 CUDA execution time for last time step iterations at a different number of implementation obtains speedup S ≈ 16. Fig. [7](#_bookmark20) presents comparing ex-

ecution time for CUDA approach and MPI with 16 processes approach implementations.



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

* 1. *Numerical results and analysis*

We used CPC-PCM to solve the hybrid fractional-order system ([9](#_bookmark8)) in order to execute the numerical simulations in the following section. We use the values of parameters in Table [2](#_bookmark10), 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

model, and *𝑁* = 11000000 [[12](#_bookmark28)] and 0 *< 𝛼* 1 and 0 *< 𝑄* 1. Fig. [8](#_bookmark21), illus- disease spread was provided. We examine the whole population in our

250

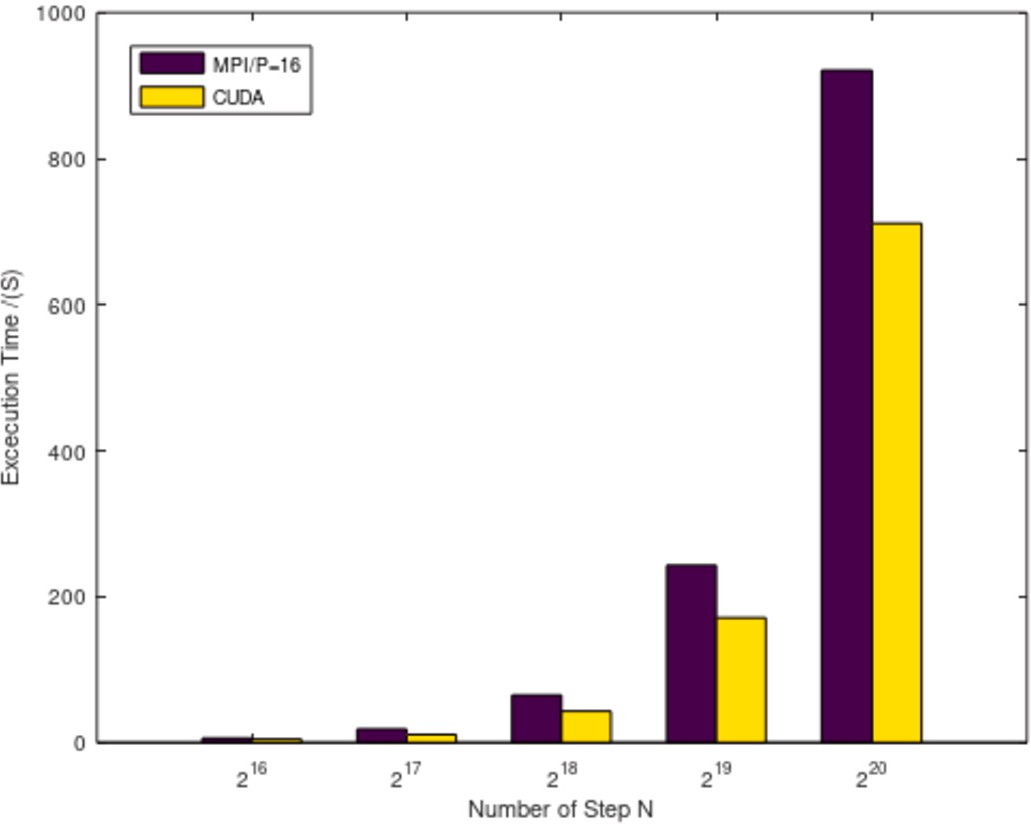
≤ ≤

different values of *𝛼* and *𝛿𝐼* = *𝛿𝑝* = *𝛿ℎ* = 0. This figure describes how so- trates the numerical answers for the suggested model’s ([9](#_bookmark8)) behaviour at lutions’ behaviour alters when we use different values *𝛼*. The fractional

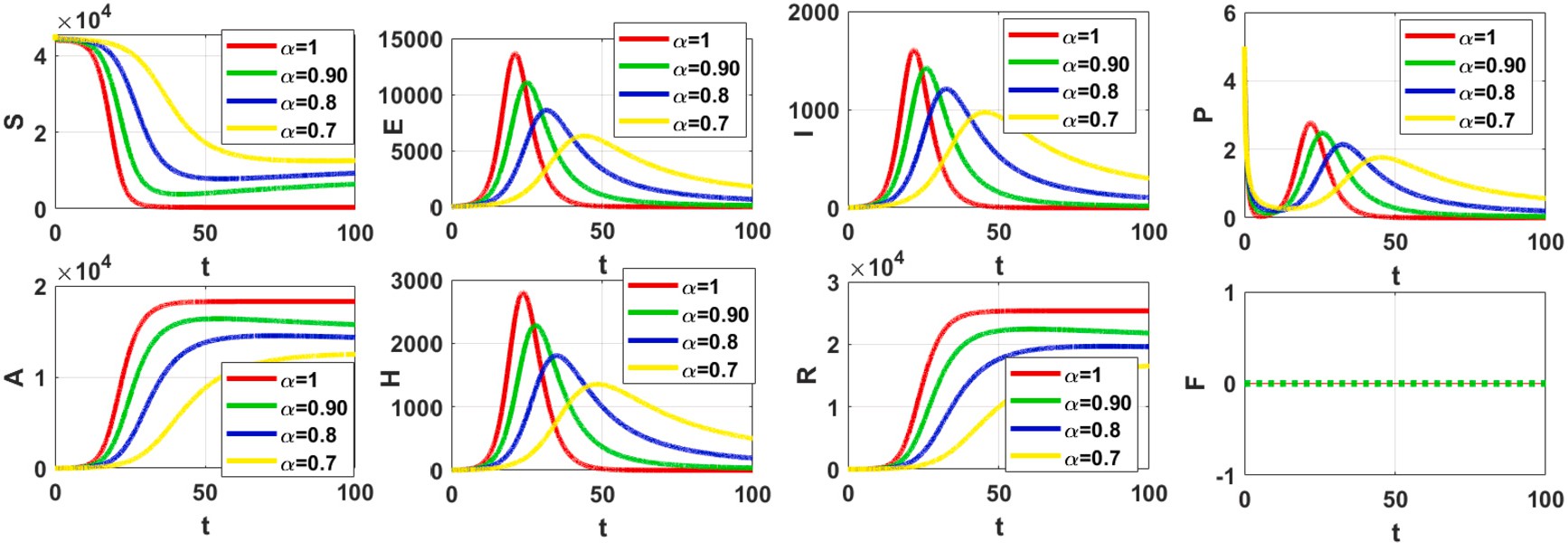
model gave us more details about spreading the disease. Moreover, this model is more general than the classical model of integer order.

# 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 us- ing 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. 7.** Execution times of CUDA and MPI 16 processes for different problem sizes N.



**Fig. 8.** The behaviour numerical solutions for the model ([9](#_bookmark8)).

nation of the integral of Riemann-Liouville and the fractional order Ca- puto derivative. Two parallel approaches are constructed based on the predictor-corrector method with the Caputo proportional constant frac- tional 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 – re- view & editing. **S. Ahmed:** Conceptualization, Data curation, Methodol- ogy, 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 mathemat- ical 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.](https://doi.org/10.1016/j.aej.2021.02.016) [aej.2021.02.016](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.](https://doi.org/10.1002/mma.8082) [org/10.1002/mma.8082](https://doi.org/10.1002/mma.8082).
4. [Yadav S, Kumar D, Singh J, Baleanu D. Analysis and dynamics of fractional order](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib53751049957164C509ABA0BE98603DA0s1) [Covid-19 model with memory effect. Results Phys 2021;24:104017.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib53751049957164C509ABA0BE98603DA0s1)
5. [Zhang L, Rahman MU, Ahmad S, Riaz MB, Jarad F. Dynamics of fractional order](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibD126FDF49369236229FDC66687B74176s1) [delay model of coronavirus disease. AIMS Math 2021;7(3):4211–32.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibD126FDF49369236229FDC66687B74176s1)
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.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib8A0813C70CF4BF1529F25C411A039130s1)
8. [Biala TA, Khaliq AQM. Parallel algorithms for nonlinear time-space fractional](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibA72D119A0C3CEB62D1C655A9CC4B7769s1) [parabolic PDEs. J Comput Phys 2018;375:135–54.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibA72D119A0C3CEB62D1C655A9CC4B7769s1)
9. [Pacheco P. An introduction to parallel programming. Burlington, NJ, USA: Morgan](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib97B47B7A63AD5387620192F8E01820D3s1) [Kaufmann; 2011.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib97B47B7A63AD5387620192F8E01820D3s1)
10. [Bonchis C, Kaslik E, Rosu F. HPC optimal parallel communication algorithm for the](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibF673265E69DDC69F280164DF249B164Cs1) [simulation of fractional-order systems. J Supercomput 2019;75:1014–25.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibF673265E69DDC69F280164DF249B164Cs1)
11. [Wang Q, Liu J, Gong C, Tang X, Fu G, Xing Z. An eﬃcient parallel algorithm for](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib43749CFD097243A97E3774723D430403s1) [Caputo fractional reaction-diffusion equation with implicit finite-difference method.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib43749CFD097243A97E3774723D430403s1) [Adv Differ Equ 2016;207.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib43749CFD097243A97E3774723D430403s1)
12. Ndärou F, Area I, Nieto JJ, Torres DFM. Mathematical modeling of COVID-19 trans- mission dynamics with a case study of Wuhan. Chaos Solitons Fractals 2020. [https://](https://doi.org/10.1016/j.chaos.2020.109846) [doi.org/10.1016/j.chaos.2020.109846](https://doi.org/10.1016/j.chaos.2020.109846).
13. Baleanu D, Fernandez A, Akgül A. On a fractional operator combining propor- tional and classical differintegrals. Mathematics 2020;8(3). [https://doi.org/10.](https://doi.org/10.3390/math8030360) [3390/math8030360](https://doi.org/10.3390/math8030360).
14. [Scherer R, Kalla S, Tang Y, Huang J. The Grünwald-Letnikov method for fractional](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib5A5942D1CF4257B457B03BD47F9F8BDCs1) [differential equations. Comput Math Appl 2011;62:902–17.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib5A5942D1CF4257B457B03BD47F9F8BDCs1)
15. [Sweilam NH, Al-Mekhlafi SM. A survey on numerical studies for fractional bio-](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib32E1EC52374360CAB7BBB777AF703B3Cs1) [logical models and their optimal control. In: Radwan Ahmad G, Khanday Farooq](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib32E1EC52374360CAB7BBB777AF703B3Cs1) [A, Said Lobna A, editors. Fractional order systems; an overview of mathematics,](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib32E1EC52374360CAB7BBB777AF703B3Cs1) [design, and applications for engineers. Academic Press. ISBN 978-0-12-824293-3,](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib32E1EC52374360CAB7BBB777AF703B3Cs1) [2022. Chapter 1.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib32E1EC52374360CAB7BBB777AF703B3Cs1)
16. [Sweilam NH, Moharram H, Moniem NKA, Ahmed S. A parallel Crank–Nicolson](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib42757B2A3060E2F64648DE1D2EBE41AAs1) [finite difference method for time-fractional parabolic equation. J Numer Math](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib42757B2A3060E2F64648DE1D2EBE41AAs1) [2014;22(4):363–82. De Gruyter.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib42757B2A3060E2F64648DE1D2EBE41AAs1)
17. [Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: a fresh approach to numerical](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib960E63ECDD393E53F9A828979F23A3B3s1) [computing. SIAM Rev 2017;59(1):65–98.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib960E63ECDD393E53F9A828979F23A3B3s1)
18. [Hunold Sascha, Steiner Sebastian. Benchmarking Julia’s communication perfor-](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibE747B90804EF98A6CBBA2832B7063C1Fs1) [mance: is Julia HPC ready or full HPC. In: 2020 IEEE/ACM performance modeling,](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibE747B90804EF98A6CBBA2832B7063C1Fs1) [benchmarking and simulation of high performance computer systems (PMBS); 2020.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibE747B90804EF98A6CBBA2832B7063C1Fs1)

[p. 20–5.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bibE747B90804EF98A6CBBA2832B7063C1Fs1)

1. [Kumar P, Rangaig NA, Abboubakar H, Kumar A, Manickam A. Prediction studies](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3B65986C758CE9B6678A51AB01316737s1) [of the epidemic peak of coronavirus disease in Japan: from Caputo derivatives to](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3B65986C758CE9B6678A51AB01316737s1) [Atangana-Baleanu derivatives. Int J Model Simul Sci Comput 2022;13(1):2250012.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3B65986C758CE9B6678A51AB01316737s1)
2. Abboubakar H, Fandio R, Sofack BS, Ekobena Fouda HP. Fractional dynam- ics of measles epidemic model. Axioms 2022;11(8):363. [https://doi.org/10.3390/](https://doi.org/10.3390/axioms11080363) [axioms11080363](https://doi.org/10.3390/axioms11080363).
3. [Kumar P, Erturk VS, Govindaraj V, Inc M, Abboubakar H, Nisar KS. Dynamics of](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib622D4C1B29C0D0C4CBE4E36C5B08195As1) [COVID-19 epidemic via two different fractional derivatives. Int J Model Simul Sci](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib622D4C1B29C0D0C4CBE4E36C5B08195As1) [Comput 2023;14(3):2350007.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib622D4C1B29C0D0C4CBE4E36C5B08195As1)
4. [Marzban HR. Optimal control of nonlinear fractional order delay systems governed](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib53073B2FCBC4534098AB2B2EB12DEBEFs1) [by Fredholm integral equations based on a new fractional derivative operator. ISA](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib53073B2FCBC4534098AB2B2EB12DEBEFs1) [Trans 2023;133:233–47.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib53073B2FCBC4534098AB2B2EB12DEBEFs1)
5. [Marzban HR, Nezami A. Analysis of nonlinear fractional optimal control systems de-](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3233F82B36D1768FC53A7E3EF42FD2ACs1) [scribed by delay Volterra-Fredholm integral equations via a new spectral collocation](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3233F82B36D1768FC53A7E3EF42FD2ACs1) [method. Chaos Solitons Fractals 2022;162:112499.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3233F82B36D1768FC53A7E3EF42FD2ACs1)
6. [Sabermahani S, Ordokhani Y, Rahimkhani P. Application of generalized Lucas](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib571DFEFFAD756BBDD8D85DE99219C854s1) [wavelet method for solving nonlinear fractal-fractional optimal control problems.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib571DFEFFAD756BBDD8D85DE99219C854s1) [Chaos Solitons Fractals 2023;170:113348.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib571DFEFFAD756BBDD8D85DE99219C854s1)
7. [Dehestani H, Ordokhani Y. An eﬃcient approach based on Legendre-Gauss-Lobatto](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3D0EE53209D10F58BB05EDC74A558C49s1) [quadrature and discrete shifted Hahn polynomials for solving Caputo-Fabrizio](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3D0EE53209D10F58BB05EDC74A558C49s1) [fractional Volterra partial integro-differential equations. J Comput Appl Math](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3D0EE53209D10F58BB05EDC74A558C49s1) [2022;403:11385.](http://refhub.elsevier.com/S1110-8665(23)00064-6/bib3D0EE53209D10F58BB05EDC74A558C49s1)