IJRCS

International Journal of Robotics and Control Systems
Vol. 6, No. 2, 2026, pp. 1452-1475
ISSN 2775-2658
http://pubs2.ascee.org/index.php/ijrcs

ASCEE

Real-Time Implementation of Nuclear Reactor Models on Embedded GPU Computing Platform: A Pathway to Hardware-in-Loop Control System

Parag R. Patil [a,1], Vishwesh A. Vyawahare [a,2,*], Sharad P. Jadhav [b,3]

[a] Department of Electronics Engineering, Ramrao Adik Institute of Technology, DY Patil Deemed to be University, Nerul, Navi Mumbai, Maharashtra 400706, India
[b] Department of Instrumentation Engineering, Ramrao Adik Institute of Technology, DY Patil Deemed to be University, Nerul, Navi Mumbai, Maharashtra 400706, India
[1] parag.patil@rait.ac.in; [2] vishwesh.vyawahare@rait.ac.in; [3] sharad.jadhav@rait.ac.in
[*] Corresponding Author

ARTICLE INFO

Article History
Received October 31, 2025
Revised December 24, 2025
Accepted May 03, 2026

Keywords
Neutron Diffusion Model (NDM);
Neutron Telegraph Model (NTM);
Hardware-in-Loop (HiL);
CUDA;
Embedded GPU;
NVIDIA Jetson Orin NX

ABSTRACT

This work presents a novel approach for parallel implementation of the neutron diffusion model (NDM) and the neutron telegraph model (NTM) (partial differential equation models) of a nuclear reactor, on an embedded Graphics Processing Unit (GPU) platform, NVIDIA Jetson Orin NX, using Compute Unified Device Architecture (CUDA). These control-oriented models are fundamental for capturing the dynamics of neutron transport in a nuclear reactor and are extensively used to design various control strategies. The computational requirements of these models are significant for real-time simulation or embedded applications. The proposed approach is based on designing algorithms for parallelizing the Separation of Variables (SoV) method for solving these models in real-time and leveraging the massively parallel architecture of the Jetson Orin's embedded GPU to accelerate their numerical solution. The embedded GPU implementation capability can significantly enhance reactor control responsiveness, support digital twin deployment, improve fault detection, and enable autonomous operation. Unlike classical GPU PDE solvers that parallelize values using neighbouring grid points (stencils), the proposed method prallelizes the full SoV directly across both spatial and temporal dimensions, incorporating warp-level modal reduction and hardware-specific mapping. The focus is on enabling efficient utilisation of GPU cores for concurrent computations, providing a high-performance computational framework. Performance evaluation is carried out by comparing the CUDA-based parallel implementation with a baseline serial implementation in C programming. Results demonstrate a substantial reduction in execution time and improved computational efficiency, while maintaining notable numerical accuracy on the embedded GPU platform. This makes the proposed approach suitable for real-time Hardware-in-Loop (HiL) implementation of the reactor models and control applications, where compact, energy-efficient, and high-performance computing is essential.

1. Introduction

Nuclear energy is a cornerstone of modern power generation, delivering large-scale, low-carbon electricity that supports energy security and sustainable development. With the rising global energy demand and the urgency of climate mitigation, nuclear power helps provide a stable, high-density energy source essential for a transition towards more resilient and clean energy systems [1]–[3].

The nuclear reactor (NR) is the core component of nuclear power generation, where controlled fission of fissile materials such as Uranium-235 or Plutonium-239 produces heat. This heat is used to generate steam, which drives turbines to produce electricity. The process is sustained by high-velocity neutrons colliding with reactor core nuclei [4]–[6]. Accurate mathematical modelling of neutron behaviour is therefore essential for designing reactors that are both efficient and safe. A nuclear fission chain reaction occurs within the reactor core when neutrons interact with the nuclei of fissile materials. The reactor consists of several key components, including the core, moderator, reflectors, fuel assemblies, control rods, and a coolant circulation system. Detailed discussions on the design and operation of nuclear reactors are available in [4]–[8].

In a nuclear reactor system, accurate reactor modelling is critical for predicting neutron behaviour, assessing reactor kinetics, and ensuring operational safety. Nuclear reactor system design, control, and emergency response planning are all based on precise simulations of neutron flux and distribution. Furthermore, safety-critical scenarios, like reactivity transients, rely on reliable predictive models to prevent accidents and guarantee stable reactor operation [6].

The neutron transport theory is based on the original neutron transport equation, which accurately characterises the distribution of neutrons in the reactor, but is difficult to solve analytically. To model neutron transport, a basic approximation called the neutron diffusion model is used [4]. The diffusion model's primary disadvantage is that it predicts an infinite neutron propagation velocity. As a result, disruption at any location in the reactor core is sensed everywhere, which is not physical given that neutrons in a reactor core have a finite velocity. This problem arises due to the parabolic nature of the diffusion equation. This setback is overcome by applying a modified constitutive law to the transport equation, resulting in the neutron telegraph model. The telegraph model, in contrast, captures finite propagation speeds and better approximates wave-like transport, offering improved fidelity especially during fast transient events [4], [9]–[11].

Nevertheless, the diffusion model plays a central role in describing neutron behaviour within nuclear reactors. Extensive research has addressed the solution of the diffusion model using both analytical and numerical approaches. Numerical studies often employ finite difference schemes or the fundamental matrix method for transient multi-group problems [12], [13], while analytical treatments can be found in [14]–[18]. Additional numerical investigations are reported in [19]–[22].

The separation of variables (SoV) method is a widely used and straightforward analytical approach for solving linear partial differential equations (PDEs). In this method, the solution is expressed as a product of separate space and time functions [23], [24]. This strategy is chosen not only for its simplicity but also because it forms the basis for developing point-kinetic models. The majority of modelling and control strategies in the case of nuclear reactor are demonstrated on 1D analytical model [4]–[6]. For real-time control, safety, robustness, and operational reliability, a 1D analytical model provides an exact solution compared to any numerical technique running on the same hardware [25].

The SoV method has been widely applied to neutron diffusion problems, including multi-group and delayed neutron models in cylindrical geometries [26], [27], fractional-order formulations [28], and general analytical treatments highlighting its simplicity and advantages. A less complicated slab geometry is considered for the reactor in this work. The key benefit of utilizing such basic geometry is that it is easy to use the one-dimensional variant of the transport equations. This simplification

makes the analysis much easier. This slab reactor geometry has been utilized by many nuclear reactor theory experts to explain fundamental and crucial concepts. The slab reactor has been quite popular in the analysis of nuclear reactors [49], [50], [62]. Solving coupled partial differential equations like the neutron diffusion and telegraph models is computationally intensive, especially when high spatial or temporal resolution, or real-time execution, is needed. Traditional serial implementations (e.g. in MATLAB or C) struggle to deliver the necessary throughput. Modern reactor safety analysis and real-time control applications demand significantly faster computation to enable timely, accurate decision-making.

Parallel computing uses multiple processors to execute tasks simultaneously, drastically reducing simulation time. In nuclear reactor modelling, data and task parallelism, for instance, across spatial grid points or time steps can be exploited to speed up numerical solutions, making real-time capable simulations feasible.

In this scenario, a Graphics Processing Unit (GPU), which is a massively parallel co-processor, can be utilised. The GPU, originally designed for graphics, is now widely used for general-purpose high-throughput computing [29]–[31]. Compute Unified Device Architecture (CUDA) is NVIDIA's parallel computing platform and programming model, enabling direct access to GPU hardware via C-like extensions. CUDA allows developers to implement compute-intensive kernels, exploit fast shared memory, and manage thousands of concurrent threads, offering higher performance gains over CPUs, especially for data-parallel workloads [32].

A literature survey reveals that there have been a few attempts to implement NR models on hardware platforms such as Field Programmable Gate Arrays (FPGAs). The work in [33] develops a 51^{st}-order nonlinear transient model of the Experimental Breeder Reactor-II (EBR-II), combining six-group neutron point kinetics with lumped-parameter heat transfer, reflector/blanket, IHX, and steam generator equations. The FPGA-based real-time digital-twin achieved 79.5% faster-than-real-time performance, accurately reproducing experimental and simulation results under various transient perturbations (e.g., reactivity insertion, feedwater temperature rise). This work presented in [34] uses a point-reactor kinetics model with one or more delayed neutron groups implemented on a Zynq SoC (FPGA + ARM) via hardware/software co-design. The hardware prototype is reported to meet real-time constraints (low latency, deterministic timing). The study in [35] developed an FPGA-based digital controller implementing the six-group point reactor kinetics equations to monitor reactivity and automatically stabilise research reactor power. Experimental validation showed reactivity and power control deviations under 5–10%, with response and accuracy comparable to the commercial system.

The hardware-in-the-loop (HiL) simulation is an advanced testing approach that combines real hardware components with virtual plant models to evaluate system performance under realistic, real-time conditions. It plays a crucial role in validating control systems for complex, safety-critical applications, such as nuclear power plants and renewable energy systems. The work in [36] verify and validates the functional performance of safety control systems, specifically a Canadian Deuterium Uranium (CANDU) nuclear power plant shutdown system by coupling real hardware, like a programmable logic controller (PLC), with a simulated plant model to ensure accurate, real-time testing before deployment. A physical Distributed Control System (DCS) and a computerized plant simulator are used in study [37] to simulate and test the real-time behaviour of nuclear power plants' major control systems utilizing HiL. The hardware controller and the software-simulated plant model for hybrid systems, such as a temperature controller in a nuclear reactor, are integrated and synchronized in the work in [38] using HiL. The work in [39] employs a real-time digital simulator (RTDS) to evaluate the power plant controller (PPC) model against its real hardware controller using HiL. For more on HiL implementations, see [40]–[42].

There have been attempts in the past to implement NR models on GPU platform; most demon-

strations are on desktop and/or server GPUs. A parallelized Adams–Bashforth– Moulton approach is used in the work in [43] to build parallel algorithms for solving nonlinear fractional-order nuclear reactor models obtained from neutron subdiffusion. Significant GPU speedups are observed across all investigated fractional differential equation models in comparison to MATLAB and CUDA implementations. The work in [44] presents optimized GPU algorithms for continuous-energy Monte Carlo neutron transport, focusing on reducing thread divergence and improving memory coalescence. It achieves significant speed-ups in particle tracking by restructuring data access patterns and refining random-number generation for GPUs.

The study in [45] demonstrates that GPUs offer massive parallelism, which accelerates the numerical solution of the diffusion equation, significantly reducing computational time compared to CPU-based methods. This enables real-time or large-scale reactor simulations with improved efficiency and scalability. A GPU-parallilized 3D neutron transport code based on the Method of Characteristics with diamond-difference (MOC/DD) and accelerated using a two-level Coarse Mesh Finite Difference (CMFD) scheme is presented in [46].

The code demonstrates high accuracy, reduced memory requirements, and significant speedups using CUDA-enabled GPUs, making whole-core transport simulations feasible on desktop-scale hardware. The GPU is utilized by implementing the discretized diffusion equation as a convolutional neural network with pre-determined weights in an AI library (TensorFlow/Keras). This allows the same code to run on GPUs without modification, exploiting the GPUs ability to perform parallel convolution operations efficiently. This approach, as reported in [47], is said to achieve approximately a 4 times speedup on GPU compared to traditional serial CPU Fortran code for the neutron diffusion model.

However, the implementation of NR models in real-time on an embedded GPU platform is a largely unexplored domain. Most existing implementations of SoV-based neutron kinetics models are offline, CPU/GPU-based, and not designed for real-time deployment. This work presents an efficient real-time implementation of nuclear reactor models on an embedded GPU platform utilizing a parallel computing architecture. The novelty of the proposed approach lies in the way the analytical SoV formulation is structured, decomposed, and parallelized on an embedded GPU for solving these models and achieving further acceleration through the compute-intensive capabilities of CUDA. The performance gain achieved by the real-time implementation of these models relative to their serial execution is also reported. The key contributions of this work can be summarized as:

1. Fully 2D parallel mapping of the SoV method for the solution of neutron diffusion and telegraph models onto CUDA grid/block/thread hierarchy.
2. Development of the SoV solution into a high-throughput, stencil-free kernel tailored to a NVIDIA Jetson Orin NX embedded GPU platform for real-time parallel computing.
3. Optimized handling of hyperbolic and oscillatory terms appearing in the neutron telegraph model.
4. Comparative analysis of the parallel CUDA implementation and a baseline serial C version, demonstrating substantial speedup and performance benchmarking.

The paper is organized as follows: Section 2 presents background concepts, detailing the diffusion and telegraph models and their mathematical foundations. Section 3 introduces GPU computing on embedded platforms. Section 4 describes the parallel algorithm design for both models under consideration. Section 5 presents experimental platforms used, computational details and the results of GPU computation using the proposed algorithms with comparison of execution time, speedups, and mean squared error between serial and parallel implementations. Section 6 provides the discussion of the results obtained. Section 7 offers conclusions and explores avenues for future research.

2. Neutron Diffusion and Telegraph Models

Nuclear reactor theory explains how neutrons behave inside a reactor core and how their interactions sustain a controlled chain reaction [3]–[5]. It covers how neutrons are produced, slowed down, absorbed, or escape, and uses mathematical models like the NDM to predict their distribution [6], [48]. While diffusion theory gives quick and useful estimates, more detailed transport theory can handle complex shapes and conditions [49], [50].

These concepts are essential for designing reactors that are safe, efficient, and reliable. The acceptance and reliability of a reactor model largely depend on how accurately it represents neutron transport within the reactor core. Therefore, an effective neutron transport model must be realistic, applicable over a wide spatial range, and computationally feasible for implementation and simulation on modern systems [28].

2.1. Neutron Diffusion Model

Conventionally, neutron motion is described using Fick's law, which leads to a parabolic partial differential equation. This equation serves as a fundamental basis for the mathematical modelling of nuclear reactors. The most common formulation is the diffusion equation, expressed in (1) as,

$$\frac{1}{v}\frac{\partial \phi(x,t)}{\partial t} + (\Sigma_a - \nu\Sigma_f)\phi(x,t) = D\frac{\partial^2 \phi(x,t)}{\partial x^2}. \tag{1}$$

Here, v denotes neutron speed, $\phi(x,t)$ represents the neutron flux at position x and time t, ν is the average number of neutrons released per fission, and D is the diffusion coefficient. Several modelling approaches, including reactor kinetics and transfer function analysis, use this parabolic PDE in (1) as their core mathematical framework.

2.2. Neutron Telegraph Model

The key limitation of the classical diffusion model is its prediction of an infinite neutron propagation speed, as discussed in [48], [51]. This means that any disturbance in the reactor core would be felt instantaneously at all other points, which is physically unrealistic. In reality, neutrons travel at a finite velocity. This issue arises from the parabolic nature of (1), which is a general shortcoming of diffusion-based formulations. To address this, Cattaneo's modified constitutive law [51]–[53] is applied. When this modification is combined with the neutron continuity equation, the neutron telegraph model is obtained as shown by (2),

$$\left(\frac{\tau}{v}\right)\frac{\partial^2 \phi(x,t)}{\partial t^2} + N_1\frac{\partial \phi(x,t)}{\partial t} + N_2\phi(x,t) = D\frac{\partial^2 \phi(x,t)}{\partial x^2}, \tag{2}$$

Where, $N_1 = \tau\Sigma_a - \tau\nu\Sigma_f + 1/v$, and $N_2 = \Sigma_a - \Sigma_f$.

The telegraph equation, given in (2), is a hyperbolic equation due to the presence of the second-order time derivative. This term introduces wave-like propagation, while the first-order term retains the diffusive component. The model naturally enforces a finite neutron propagation speed, calculated as $v = 3D/\tau$. In scenarios where the absorption cross section is much larger than the transport cross section, the telegraph model provides more accurate results than the diffusion model [48]. In the special case where $\tau \to 0$, the telegraph model reduces to the standard diffusion model (1).

2.3. Separation of Variables Method

The NDM and NTM can be solved using the separation of variables method [4]. This classical analytical approach for the solution of linear PDEs assumes that the neutron flux can be written as a

product of space and time-dependent functions as expressed in (3) as,

$$\phi(x,t) = \psi(x) \cdot T(t), \tag{3}$$

Where, $\psi(x)$ and $T(t)$ satisfy the corresponding space and time ordinary differential equations (ODEs). This work considers a uniform slab geometry of fissile material of length 'a' with absorption, transport, and fission cross sections Σ_a, Σ_{tr}, and Σ_f, respectively. The initial and boundary conditions are defined as follows:

Initial Condition (IC):

The initial flux distribution is assumed to be symmetric as,

$$\phi(x,0) = \phi_0(x) = \phi_0(-x), \tag{4}$$

with $\phi_0(x)$ taken as a triangular distribution as shown in Fig. 1.

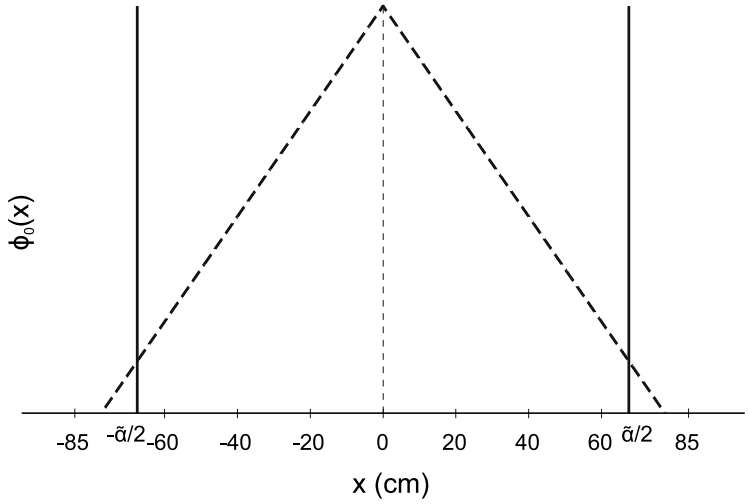


Fig. 1. Initial distribution of flux: $\phi_0(x)$

Boundary Condition (BC):

Flux vanishes at the extrapolated boundaries,

$$\phi\left(\pm\tfrac{\tilde{a}}{2}, t\right) = 0, \tag{5}$$

Where, $\tilde{a} = a + z_0$ and $z_0 = 0.71\lambda_{tr}$ is the extrapolation distance. Applying SoV yields separate spatial and temporal equations. The complete solution is expressed as an infinite series of eigenfunctions as,

$$\phi(x,t) = \sum_{n \text{ odd}} T_n(t) \cdot \psi_n(x). \tag{6}$$

The following subsections outline the SoV solutions for the diffusion and telegraph models using ICs and BCs mentioned by (4) and (5) respectively, with emphasis on their computational complexity.

2.3.1. Solution of Neutron Diffusion Model using SoV Method

The solution of NDM (1) can be expressed as an infinite sum of products of $T_n(t)$ and $\psi_n(x)$, as shown in (6).

For the NDM model:

1. Space ODE

$$D\frac{d^2\psi(x)}{dx^2} + (\nu\Sigma_f - \Sigma_a)\psi(x) = -\frac{\lambda}{v}\psi(x). \tag{7}$$

2. Time ODE

$$\frac{dT(t)}{dt} = -\lambda T(t). \tag{8}$$

Solution of the Space ODE:

$$\psi_n(x) = \cos(B_n x), \tag{9}$$

Where,

$$B_n^2 = \left(\frac{n\pi}{\tilde{a}}\right)^2, \quad n = 1, 3, 5, \ldots \tag{10}$$

Solution of the Time ODE:

$$T_n(t) = T_n(0)\, e^{-\lambda_n t}, \tag{11}$$

Where,

$$T_n(0) = \frac{16}{n^2\pi^2}\sin^2\left(\frac{n\pi}{4}\right), \tag{12}$$

and,

$$\lambda_n = v\Sigma_a + vDB_n^2 - v\nu\Sigma_f. \tag{13}$$

Here, λ is a constant, and IC $T_n(0)$ is the same for NDM and NTM presented in this work. Thus, by substituting (9), (11) into (6), the neutron flux profile for NDM in the slab reactor can be obtained.

2.3.2. Solution of Neutron Telegraph Model using SoV Method

The NTM given in (2) can also be solved using the separation of variables method, resulting in the same general form as (6). For the NTM model:

1. Space ODE

$$\frac{d^2\psi(x)}{dx^2} + \frac{1}{D}\left[\nu\Sigma_f - \Sigma_a + \frac{\tau}{v}\lambda\right]\psi(x) = 0, \tag{14}$$

2. Time ODE

$$\frac{d^2T(t)}{dt^2} + N_{11}\frac{dT(t)}{dt} + \lambda T(t) = 0. \tag{15}$$

Since the same boundary and initial conditions are used, the space ODE solution is the same as (9). **Solution of the Time ODE**:

$$T_n(t) = T_n(0)\, e^{-N_{11}t/2}\left[\frac{N_{11}\sinh\left(\frac{t}{2}\sqrt{N_{11}^2 - 4\lambda_n}\right)}{\sqrt{N_{11}^2 - 4\lambda_n}} + \cosh\left(\frac{t}{2}\sqrt{N_{11}^2 - 4\lambda_n}\right)\right], \tag{16}$$

Where, $T_n(0)$ is given by (12),

$$N_{11} = \frac{v}{\tau}N_1, \tag{17}$$

and,

$$\lambda_n = \frac{v}{\tau}\left[\Sigma_a - \nu\Sigma_f + DB_n^2\right]. \tag{18}$$

By substituting (9), (16) into (6), the distribution of neutron flux in the reactor core can be modelled. Although the SoV method yields exact analytical solutions for the neutron diffusion and telegraph models, it requires evaluating an infinite series of spatial and temporal nodes. In practice, accurate results demand summing a large number of terms, each involving trigonometric or exponential functions as demonstrated by (6), (9), (11), and (16). The cost grows quickly with finer spatial resolution, complex boundary conditions, or multi-group extensions [14], [16].

As a result, SoV becomes computationally expensive for realistic reactor models, making it a strong candidate for GPU-based parallelization with CUDA, where each node or spatial point can be processed concurrently. Hence, parallel computing with CUDA can be employed to distribute the workload of evaluating these terms across multiple GPU cores. This greatly accelerates computation while maintaining the accuracy of the SoV method-based solutions. The further section explains what GPU computing is, and with reference to literature, lists applications of GPU computing using embedded platforms in various domains.

3. GPU Computing using Embedded Platforms

Graphics Processing Unit computing refers to the utilization of a GPU as a co-processor to accelerate computationally intensive workloads traditionally executed on the central processing unit (CPU). While CPUs are designed with a small number of complex cores optimized for sequential instruction execution, GPUs contain thousands of simpler, lightweight cores optimized for massively parallel processing [30], [54]. This architectural difference enables GPUs to achieve significantly higher throughput for algorithms that can be decomposed into parallel tasks. By exploiting data-level parallelism, GPUs can accelerate simulations that would otherwise require prohibitive computation times on conventional CPUs.

Embedded GPU platforms extend this capability to low-power, compact environments. Devices such as the NVIDIA Jetson Orin integrate ARM-based CPUs with CUDA-enabled GPUs in a single System-on-Chip (SoC), enabling high-performance parallel computing [55]. Several studies have shown the effectiveness of embedded GPUs in accelerating real-time applications. For example, GPU-accelerated embedded systems have been used for deep learning-based object detection [56], real-time control in robotics [57], and medical image processing [58]. These works report significant performance improvements compared to CPU-only systems, validating the role of embedded GPUs in time-critical computations.

The work presented in [59] highlights that the Jetson Orin Nano offers significant benefits for running vision transformer inference, achieving 6 to 32 times faster inference speeds compared to CPU execution while consuming 6 to 10 times less energy, making it ideal for real-time AI applications in resource-constrained environments. The contributions in [60] evaluate thermal object detection for automotive ADAS using YOLO-v5 networks deployed on embedded GPU platforms, achieving a 3.5 times speedup.

This enables low-power on-board deployment for vehicular systems while maintaining detection accuracy across challenging weather conditions, where thermal imaging outperforms visible-light cameras. The work in [61] reports the comparison of two embedded low-power GPU platforms (NVIDIA Jetson Nano and Qualcomm Snapdragon with Adreno GPU) for real-time traffic surveillance. The embedded GPUs enable real-time processing of high definition video streams, achieving 16-34 fps for different resolutions while providing massive speedups (43 to 477 times) over sequential CPU implementations and outperforming desktop OpenCV implementations.

To harness this computational power, NVIDIA provides the Compute Unified Device Architecture

framework, which enables developers to write GPU-accelerated programs in a C/C++-like syntax. Unlike MATLAB or traditional serial C implementations, CUDA offers direct control over thread organization, memory hierarchy, and device-specific optimizations, enabling higher efficiency for large-scale parallel workloads [54]. In the context of solving the NDM and NTM, CUDA facilitates data-parallel execution, drastically reducing runtime while maintaining numerical accuracy.

4. Parallel Algorithm Design for Solution of NDM and NTM using SoV

The neutron diffusion and neutron telegraph model involve solving large systems of coupled differential equations across discrete spatial and temporal grids. These computations are inherently parallelizable because the value of ϕ at each grid point (x, t) depends primarily on the solution of space ODE and time ODE as explained in Subsection 2.3, which are not dependent. Such data parallelism makes these models ideal candidates for GPU acceleration using CUDA. The CUDA implementation of NDM and NTM using SoV utilizes the following guidelines:

- **Domain Decomposition**: The spatial domain is mapped to a 2D CUDA grid, where each thread computes the solution for a single spatial cell at a given time step.
- **Thread Organization**: Threads are arranged in `dim3` blocks and grids, ensuring memory coalescing for neighbour access in stencil computations.
- **Memory Hierarchy Optimization**: Frequently accessed coefficients and constants are stored in constant memory, while intermediate values use shared memory within thread blocks to reduce global memory access latency.
- **Overlapping Computation and Data Transfer**: CUDA streams are employed to overlap kernel execution with host-to-device and device-to-host memory transfers.
- **Algorithm-Specific Kernel Design**: Separate kernels are written for NDM and NTM, reflecting differences in their temporal update schemes.

In the neutron diffusion model, the spatial variation of flux is governed by second-order derivatives. Since exact solutions are not always feasible, these derivatives are approximated numerically using the finite-difference method (FDM), where the reactor core is divided into discrete spatial grid points and derivatives are replaced by difference equations. The neutron flux is updated by an implicit time-marching scheme, where the flux at the next time step depends on itself, leading to a system of equations that must be solved at each step. It is more stable, but computationally more expensive, as demonstrated by Algorithm 1, which shows the host implementation, and Algorithm 2, which shows the CUDA kernel for parallel implementation on the GPU.

The neutron telegraph model includes a second-order time derivative, and the kernel described by Algorithm 4 incorporates additional terms involving hyperbolic or oscillatory branches (depending on the discriminant). The branching avoids significant thread divergence, because the condition is driven solely by k (the loop index) and is identical across all threads for a given loop iteration. Each warp is executing the same k at the same time, so the branch is uniform per-warp. Hence, the warp-level sorting is unnecessary in this case. This kernel-level customization allows the GPU to handle each model's specific mathematical behaviour efficiently, maintaining numerical accuracy while exploiting maximum parallelism. Each kernel is optimized by adjusting register usage, shared memory size, and thread configuration. The host implementation for NTM is described by Algorithm 3, and kernel launch on the device is described by Algorithm 4.

The proposed SoV approach is inherently stable as it employs full parallelization as shown in Fig. 2, avoiding time-stepping instability issues occurring in many PDE solvers. This scales well in a parallel execution scenario because all threads are independent.

Algorithms for NDM and NTM both achieve significant acceleration by:

1. Exploiting fine-grained parallelism over spatial and temporal grid points.

2. Using shared memory for cooperative data sharing among multiple threads in a block, thus reducing global memory access.
3. Using constant memory to reduce memory latency, global memory bandwidth usage, and device memory allocations.
4. Minimizing host-device data transfers inside the main time loop.

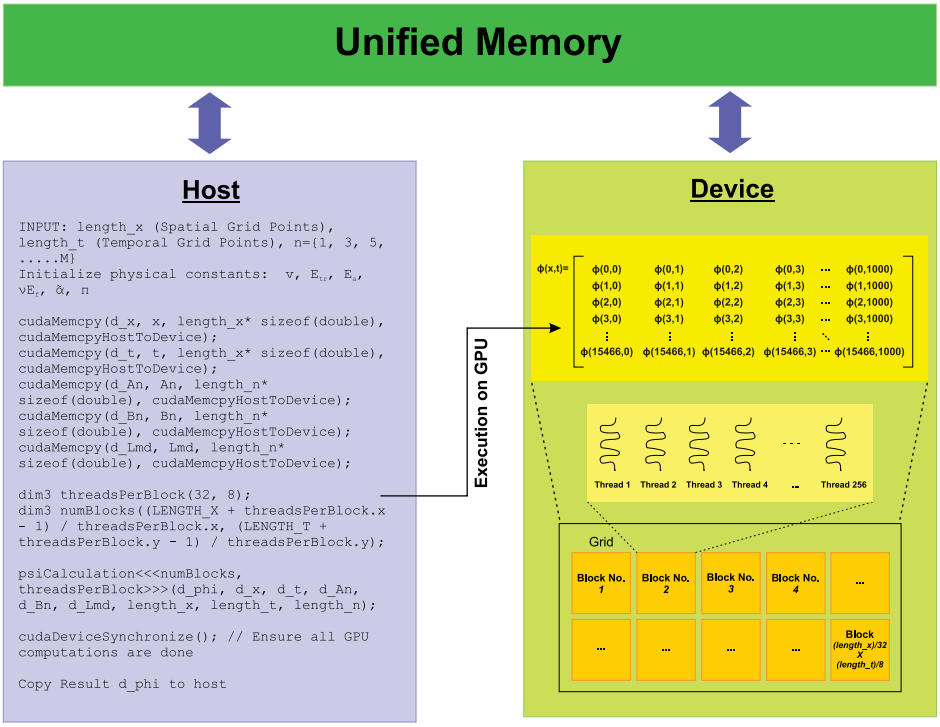


Fig. 2. Pictorial representation of mapping utilised by the proposed algorithm kernel

The GPU computation of these proposed algorithms and the results obtained are presented in Section 5.

Algorithm 1 Calculation of $\phi(x,t)$ for NDM (Host Implementation)

Require: $M \geq 1$, discretization parameters for x and t
Ensure: $\phi[length\_x \times length\_t]$
Initialize physical constants: $v, E_{tr}, E_a, \nu E_f, \tilde{a}, \pi$
Generate odd indices: $n = \{1, 3, 5, \ldots, M\}$
$D \leftarrow 1/(3 \cdot E_{tr})$
for $i = 0$ to $(length\_n - 1)$ **do**
    $B_n[i] \leftarrow (\pi/\tilde{a}) \cdot n[i]$
    $A_n[i] \leftarrow \frac{16}{n[i]^2 \pi^2} \sin^2\left(\frac{n[i]\pi}{4}\right)$
    $\lambda[i] \leftarrow v \cdot (E_a + D \cdot B_n[i]^2 - \nu E_f)$
end for
Discretize $x \in [-\tilde{a}/2, \tilde{a}/2]$ with $\Delta x = 0.01$
Discretize $t \in [0, 0.001]$ with $\Delta t = 10^{-6}$
Allocate GPU memory for ϕ, x, t
Copy x, t to device memory
Copy A_n, B_n, λ to constant memory
Configure kernel launch:
Configure launch: $block = (32, 8)$, $grid = (\lceil length\_x/32 \rceil, \lceil length\_t/8 \rceil)$
Launch kernel: PHICALCULATION$(d_phi, d_x, d_t, length_x, length_t, length_n)$
Copy result $d\_phi \rightarrow \phi$
Save ϕ to CSV file
return ϕ

Algorithm 2 CUDA Kernel phiCalculation() Launch for NDM (Device Implementation)

Require: $d\_phi, d\_x, d\_t, length\_x, length\_t, length\_n$
Ensure: $\phi[k_1, k_2]$ for each thread (k_1, k_2)
$k_1 \leftarrow blockIdx.x \cdot blockDim.x + threadIdx.x$ ▷ x index
$k_2 \leftarrow blockIdx.y \cdot blockDim.y + threadIdx.y$ ▷ t index
Shared memory: $sh\_x[32], sh\_t[8]$
if $threadIdx.y = 0$ and $k_1 < length\_x$ **then**
  $sh\_x[threadIdx.x] \leftarrow d\_x[k_1]$
end if
if $threadIdx.x = 0$ and $k_2 < length\_t$ **then**
  $sh\_t[threadIdx.y] \leftarrow d\_t[k_2]$
end if
__SYNCTHREADS
if $k_1 < length\_x$ and $k_2 < length\_t$ **then**
  $sm \leftarrow 0.0$
  $local\_x \leftarrow sh\_x[threadIdx.x]$
  $local\_t \leftarrow sh\_t[threadIdx.y]$
  for $k_3 = 0$ to $(length\_n - 1)$ **do**
    $sm \leftarrow sm + d\_A_n[k_3] \cdot \exp(-d\_\lambda[k_3] \cdot local\_t) \cdot \cos(d\_B_n[k_3] \cdot local\_x)$
  end for
  $d\_phi[k_1 \cdot length\_t + k_2] \leftarrow sm$
end if

Algorithm 3 Calculation of $\phi(x, t)$ for NTM (Host Implementation).

Require: Physical constants $v, \Sigma_{tr}, \Sigma_a, \nu\Sigma_f, \tilde{a}$; discretization $dx, dt, t_{\text{start}}, t_{\text{end}}$; mode count M
Ensure: $\phi \in \mathbb{R}^{N_x \times N_t}$
Precompute constants:
  $\tau \leftarrow 1/(v \cdot \Sigma_{tr})$
  $D \leftarrow 1/(3 \cdot \Sigma_{tr})$
  $N_1 \leftarrow \tau(\Sigma_a - \nu\Sigma_f) + 1/v$
  $N_{11} \leftarrow (v/\tau)\, N_1$
  $K \leftarrow (M + 1)/2$ ▷ number of odd modes
Build grids:
  $x\_array \leftarrow \text{linspace}(-\tilde{a}/2,\ \tilde{a}/2,\ N_x)$
  $t\_array \leftarrow \text{linspace}(t_{\text{start}},\ t_{\text{end}},\ N_t)$
Compute mode coefficients on host (for $k = 0 \ldots K - 1$**):**
for $k = 0$ to $K - 1$ **do**
  $n \leftarrow 2k + 1$ ▷ odd mode index
  $B[k] \leftarrow (n\pi)/\tilde{a}$
  $T0[k] \leftarrow \dfrac{16}{n^2\pi^2} \sin^2\left(\dfrac{n\pi}{4}\right)$
  $\lambda_k \leftarrow (v/\tau)\left(\Sigma_a + DB[k]^2 - \nu\Sigma_f\right)$
  $disc \leftarrow N_{11}^2 - 4\lambda_k$
  if $disc \geq 0$ **then**
    $term2[k] \leftarrow \sqrt{disc};\ isOsc[k] \leftarrow 0$ ▷ hyperbolic branch
  else
    $term2[k] \leftarrow \sqrt{-disc};\ isOsc[k] \leftarrow 1$ ▷ oscillatory branch
  end if
end for
Upload to device: copy x_array, t_array to d_x, d_t; copy $B, T0, term2, isOsc, N_{11}$ to device constant memory
Allocate device array $d\_\phi[N_x \times N_t]$
Configure launch: $block = (32, 8)$, $grid = (\lceil N_x/32 \rceil, \lceil N_t/8 \rceil)$
Launch kernel: PHICALCULATION$(d_\phi, d_x, d_t, N_x, N_t, K)$
Copy back: $d\_\phi \rightarrow \phi$
Store ϕ (CSV / plots)
return ϕ

Algorithm 4 CUDA Kernel phiCalculation() Launch for NTM (Device Implementation)

Require: device arrays $d\_\phi, d\_x, d\_t, length\_x, length\_t$, number of modes K
Ensure: each thread computes one entry $\phi[i, j]$

$i \leftarrow blockIdx.x \cdot blockDim.x + threadIdx.x$ ▷ spatial index
$j \leftarrow blockIdx.y \cdot blockDim.y + threadIdx.y$ ▷ temporal index
shared $sh\_x[32]$ $sh\_t[8]$ ▷ match blockDim
if threadIdx.y == 0 **and** i ¡ N_x **then**
  $sh\_x[threadIdx.x] \leftarrow d\_x[i]$
end if
if threadIdx.x == 0 **and** j ¡ N_t **then**
  $sh\_t[threadIdx.y] \leftarrow d\_t[j]$
end if
__SYNCTHREADS
if $i \geq length\_x$**or**j $\geq length\_t$ **then**
  return
end if
$x_i \leftarrow sh\_x[threadIdx.x]$; $t_j \leftarrow sh\_t[threadIdx.y]$
$expfac \leftarrow \exp(-N_{11}\, t_j/2)$
$sum \leftarrow 0.0$
for $k = 0$ to $K - 1$ **do** ▷ mode summation (parallel over (i, j))
  if $isOsc[k] == 0$ **then** ▷ hyperbolic case
    $half \leftarrow 0.5 \cdot t_j \cdot term2[k]$
    $T_k \leftarrow T0[k] \cdot expfac \cdot \left(\frac{N_{11} \sinh(half)}{term2[k]} + \cosh(half)\right)$
  else ▷ oscillatory case (complex roots)
    $half \leftarrow 0.5 \cdot t_j \cdot term2[k]$
    $T_k \leftarrow T0[k] \cdot expfac \cdot \left(\frac{N_{11} \sin(half)}{term2[k]} + \cos(half)\right)$
  end if
  $sum \leftarrow sum + T_k \cdot \cos(B[k] \cdot x_i)$
end for
$d\_\phi[i \cdot length\_t + j] \leftarrow sum$

5. GPU Computation of the Proposed Algorithms

To accelerate the solution of the nuclear reactor models, the proposed algorithms are implemented on a GPU using NVIDIA's CUDA framework. Unlike sequential CPU execution, where computations are performed one after another, CUDA enables thousands of threads to run in parallel, making it highly effective for handling the spatial and temporal evaluations required in the SoV method. By exploiting the parallel architecture of GPU, the computation time is drastically reduced, enabling faster and more efficient simulation of neutron flux evolution in reactor core. This section details the experimental platform and performance results of the proposed parallel algorithms and speedup achieved.

5.1. Experimental Platform

The NVIDIA Jetson Orin NX is a compact, high-performance edge computing module designed for AI, robotics, and real-time embedded applications. It combines an Arm® Cortex®-A78AE multi-core CPU with an Ampere-based GPU and is optimized for low-power, high-throughput scenarios. With configurable power modes ranging from 10 W to 25 W, the Jetson Orin NX delivers up to 100 INT8 TOPS of AI performance while maintaining energy efficiency and a small form factor, making it ideal for embedded systems that demand both performance and portability.

The Fig. 3 shows the experimental setup utilized, which comprises NVIDIA Jetson Orin NX for embedded computing, Digilent Analog Discovery 3 for digital to analogue conversion of flux values, and a digital storage oscilloscope (DSO) for capturing the flux behaviour. Such experimental visualisation is significant because it bridges the gap between theoretical reactor models and their

real-time deployment in monitoring and control systems, where rapid response and reliability are essential for ensuring reactor safety and stability. The following Table 1 highlights the specifications of the onboard CPU and GPU of Jetson platform utilized for the computation.

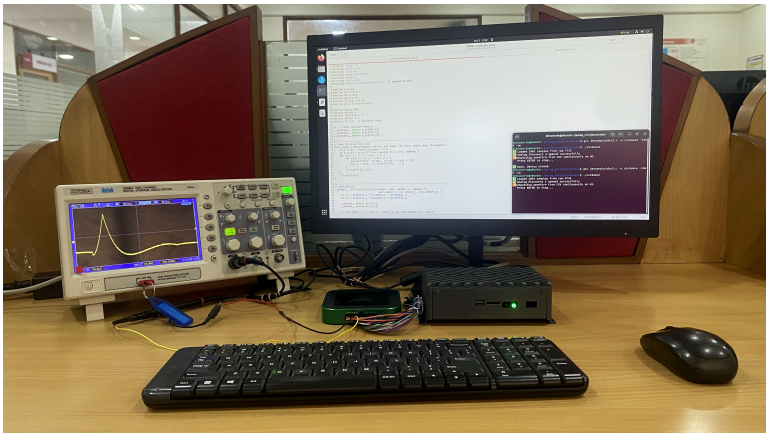


Fig. 3. Experimental setup with jetson orin NX

Table 1. Specifications of NVIDIA Jetson Orin NX (CPU & GPU)

Component	Specification
Onboard CPU	
Architecture	8-core Arm® Cortex®-A78AE v8.2 64-bit CPU
Clock Speed	Up to 2.0 GHz
L2 + L3 Cache	2 MB L2 + 4 MB L3
Instruction Set	ARMv8.2 with SIMD and FP support
Onboard GPU	
Architecture	NVIDIA Ampere Architecture
CUDA Cores	1024 cores
Clock Speed	Up to 918 MHz
Shared Memory / L1 Cache	128 KB per SM

5.2. Computational Parameters

The two nuclear reactor models, NDM and NTM are solved using SoV method for slab geometry with model parameters and ICs, and BCs as mentioned in Section 2.3. The computational parameters used in this study are as follows:

1. The space grid is $x = [-77.29 : 0.01 : 77.29]$.
2. The time vector is $t = [0 : 0.000001 : 0.001]$.
3. The summation in (6) is carried out up to $n = 1001$.

The validity of the results obtained using the proposed parallel algorithms on the onboard NVIDIA GPU of Jetson Orin NX is investigated by comparing the results with those of the sequential C codes executed on the onboard CPU and the sequential MATLAB codes executed on a desktop system.

5.3. Results

The SoV method for the solution of NDM is implemented on the GPU using CUDA, as outlined by Algorithm 1 and Algorithm 2 of Section 4. The time evolution of the neutron flux, $\phi(x,t)$, at spatial locations $x = 0$, 20, 40, and 60 (cm) is presented in Fig. 4.

The results show that the flux gradually stabilizes to a steady value, exhibiting an overall exponential trend. The spatial distribution of neutron flux within the slab geometry at time instants $t = 0$, 2.5×10^{-5}, 5.0×10^{-5}, and 1.0×10^{-3} (s) is depicted in Fig. 5. As observed, the flux progressively

evolves toward a steady-state cosine-shaped spatial distribution from an initial triangular shape shown by Fig. 1.

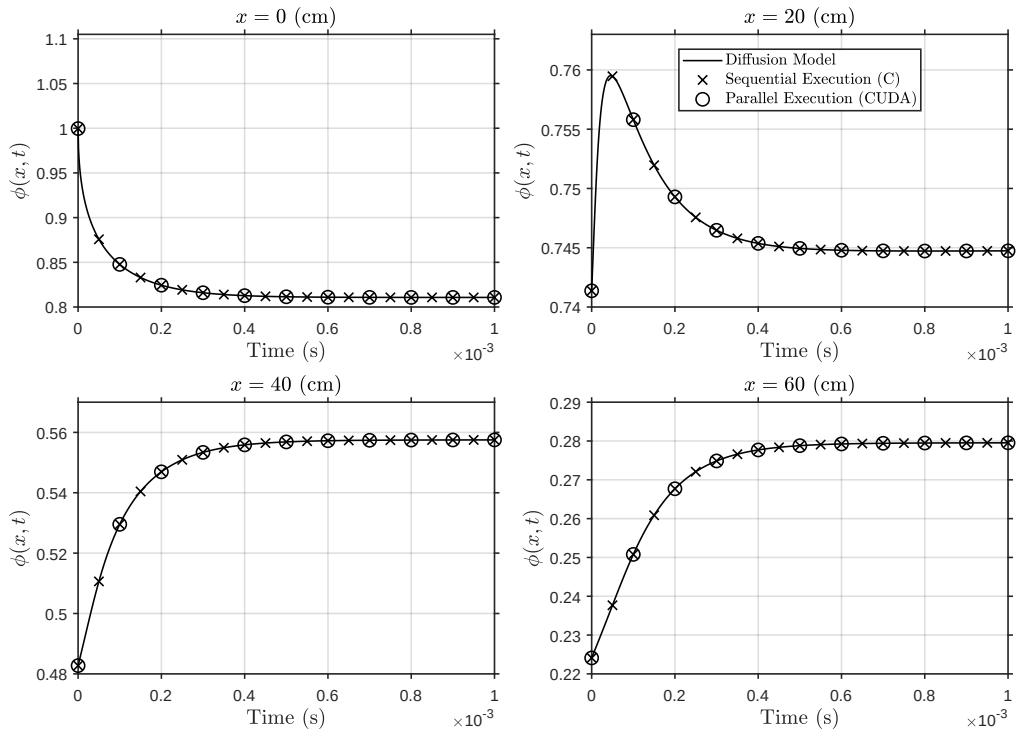


Fig. 4. Benchmarking of time-evolution of $\phi(x,t)$ for NDM at $x = 0, 20, 40$, and 60 (cm)

The SoV method for the solution of NTM is implemented on the GPU using the CUDA framework, as given in Algorithm 3 and Algorithm 4 in Section 4. The flux evolution $\phi(x,t)$ at spatial locations $x = 0$, 20, 40, and 60 (cm) is presented in Fig. 6. The results show that, unlike the NDM case, for NTM the flux evolution is initially non-exponential but eventually stabilizes to a steady value similar to that observed for NDM.

Fig. 7 illustrate the spatial distribution of neutron flux at time instants $t = 0$, 2.5×10^{-5}, 5.0×10^{-5}, and 1.0×10^{-3} (s) . It can be observed that the flux gradually evolves and eventually attains a cosine-shaped spatial profile similar to NDM.

To validate the practicality of the proposed models and their GPU-based acceleration for hardware implementation, the results were captured on a digital storage oscilloscope for both the NDM and NTM models. This hardware-based demonstration highlights not only the correctness of the computed solutions but also their real-time feasibility when implemented on embedded GPU platforms.

The Fig. 8 and Fig. 10 showcase the time evolution of flux $\phi(x,t)$ at selected space points for NDM and NTM, respectively, on DSO. Similarly, the Fig. 9 and Fig. 11 depict the spatial evolution of flux $\phi(x,t)$ at selected time intervals for NDM and NTM, respectively, on DSO.

On comparing these Fig. 8, Fig. 9, Fig. 10, and Fig. 11 with Fig. 4, Fig. 5, Fig. 6, and Fig. 7 respectively, the results obtained by hardware-based demonstration can be validated. Table 2 presents a comparison of the execution times for the neutron diffusion and the neutron telegraph Model obtained using the traditional sequential CPU implementation in C and the proposed parallel CUDA-based embedded GPU implementation. The table also reports the performance speed-up achieved by the GPU execution relative to the CPU execution.

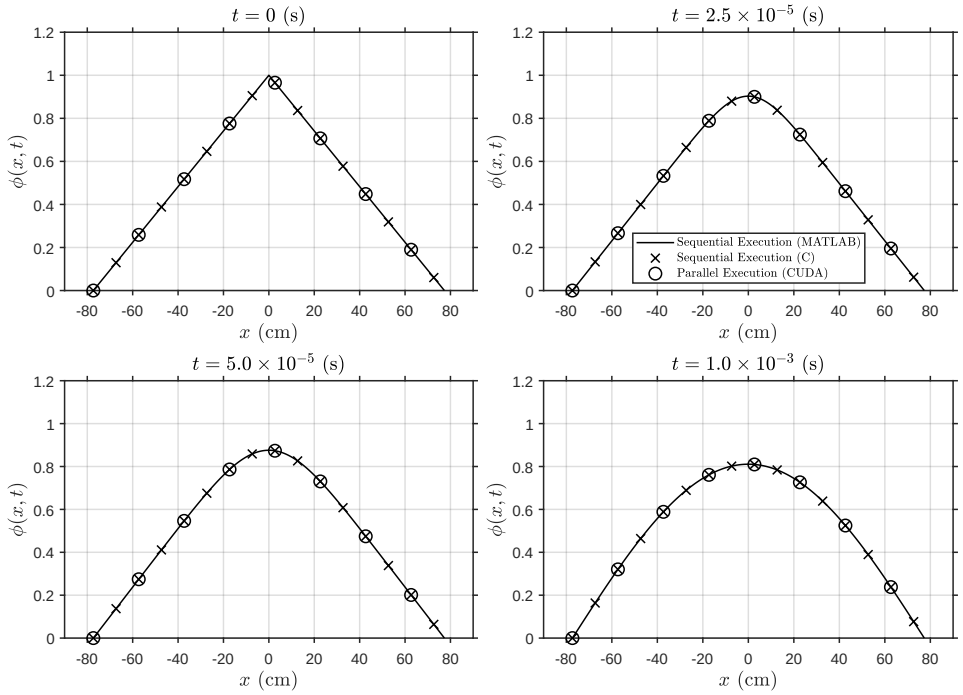


Fig. 5. Benchmarking of spatial evolution of $\phi(x,t)$ for NDM at $t = 0, 2.5 \times 10^{-5}, 5.0 \times 10^{-5}$ and 1.0×10^{-3} (s)

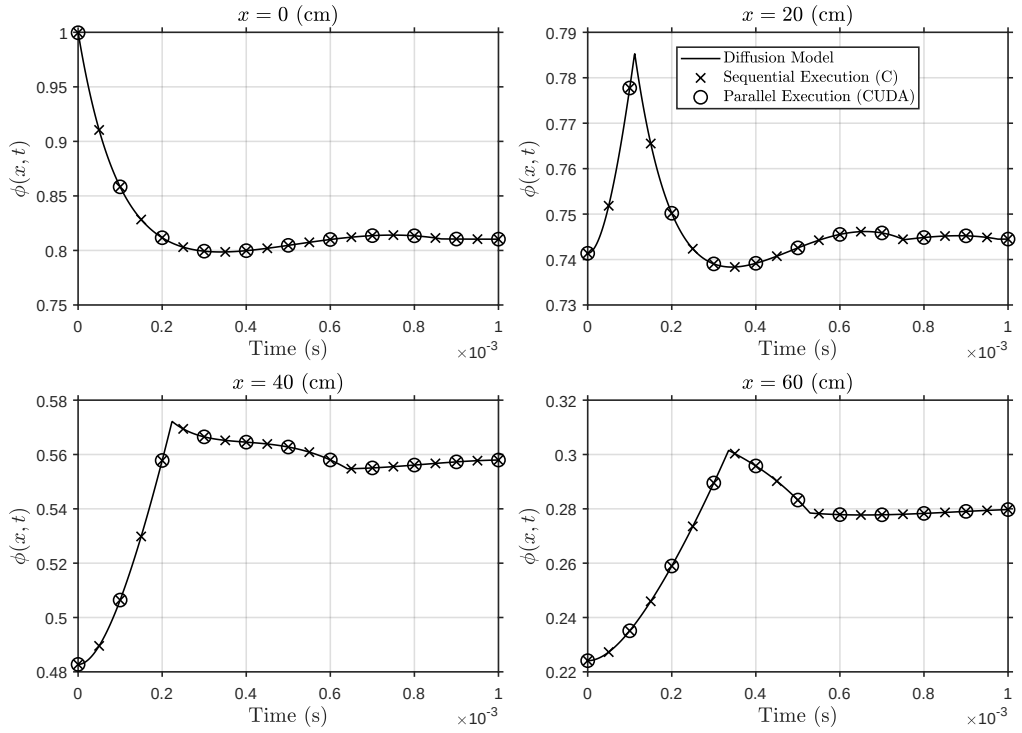


Fig. 6. Benchmarking of time-evolution of $\phi(x,t)$ for NTM at $x = 0, 20, 40$, and 60 (cm)

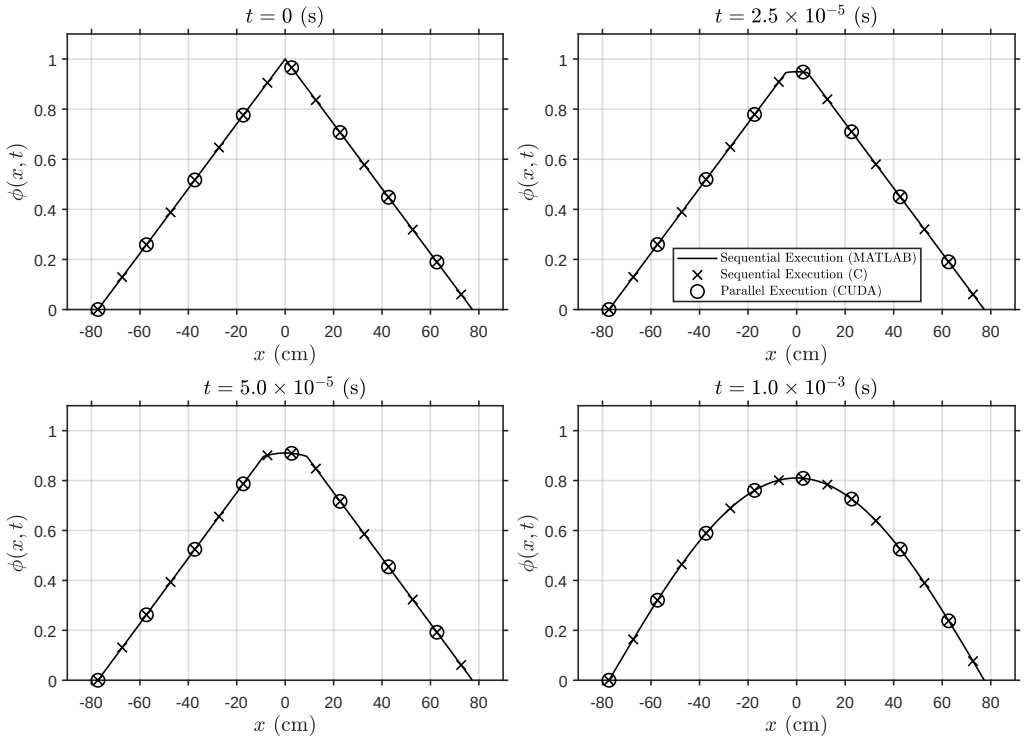
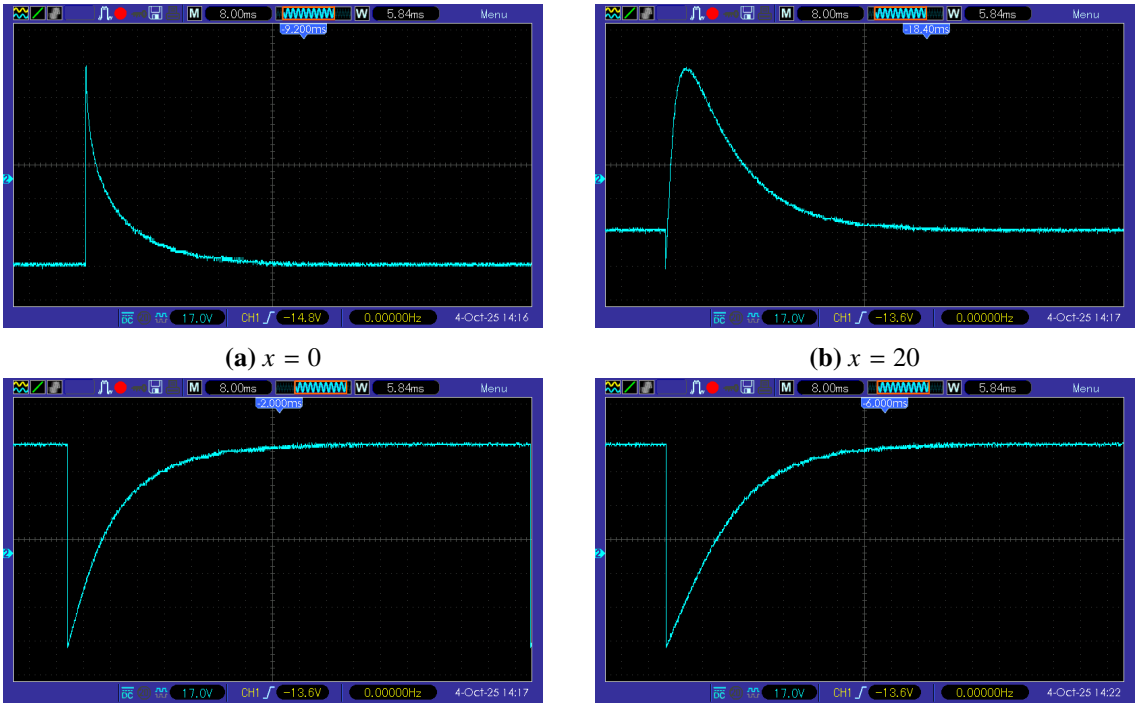


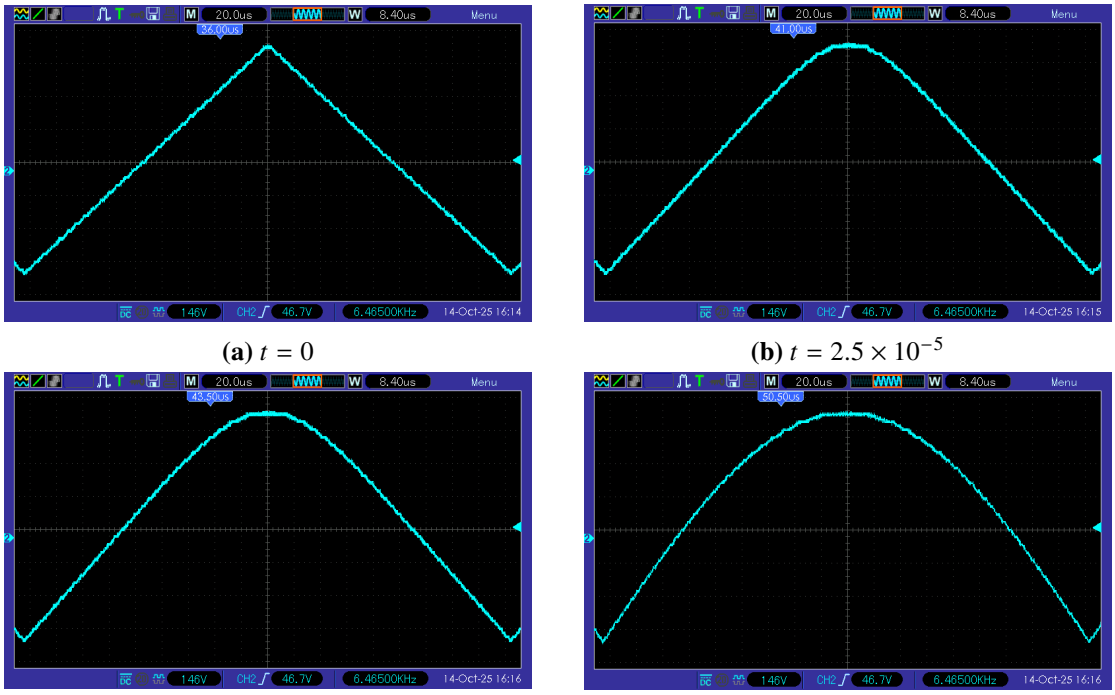
Fig. 7. Benchmarking of spatial evolution of $\phi(x,t)$ for NTM at $t = 0, 0.25 \times 10^{-3}$, and 1.0×10^{-3} (s)



(a) $x = 0$ **(b)** $x = 20$

(c) $x = 40$ **(d)** $x = 60$

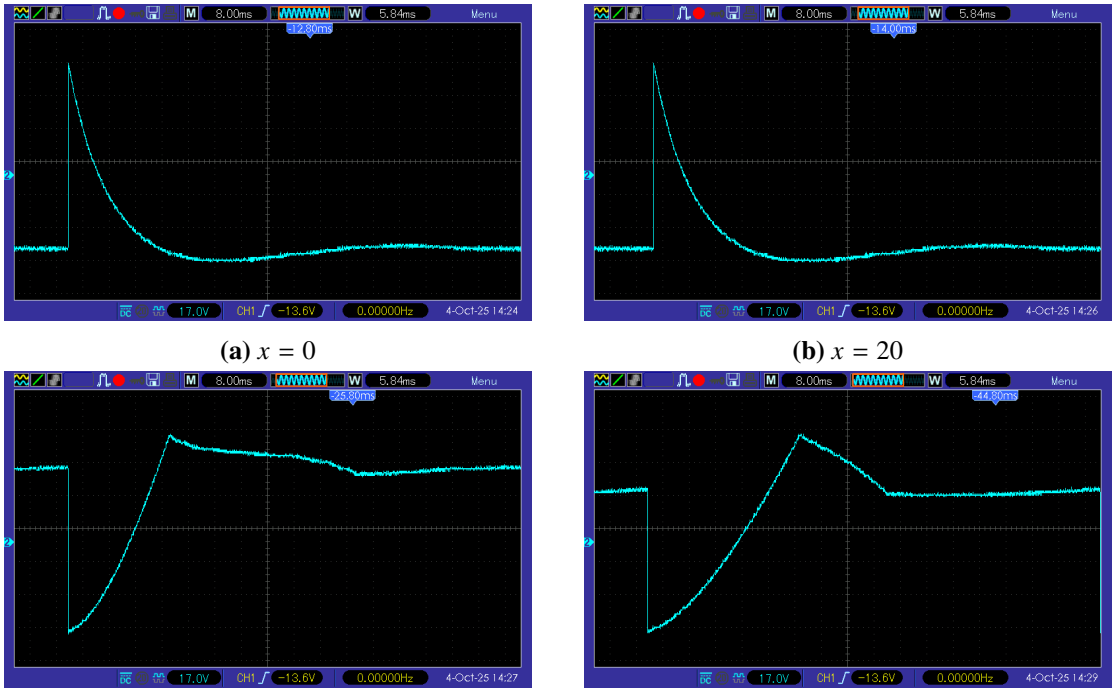
Fig. 8. Time-evolution of $\phi(x,t)$ for NDM displayed on DSO



(a) $t = 0$ **(b)** $t = 2.5 \times 10^{-5}$

(c) $t = 5.0 \times 10^{-5}$ **(d)** $t = 1.0 \times 10^{-3}$

Fig. 9. Spatial Evolution of $\phi(x, t)$ for NDM displayed on DSO

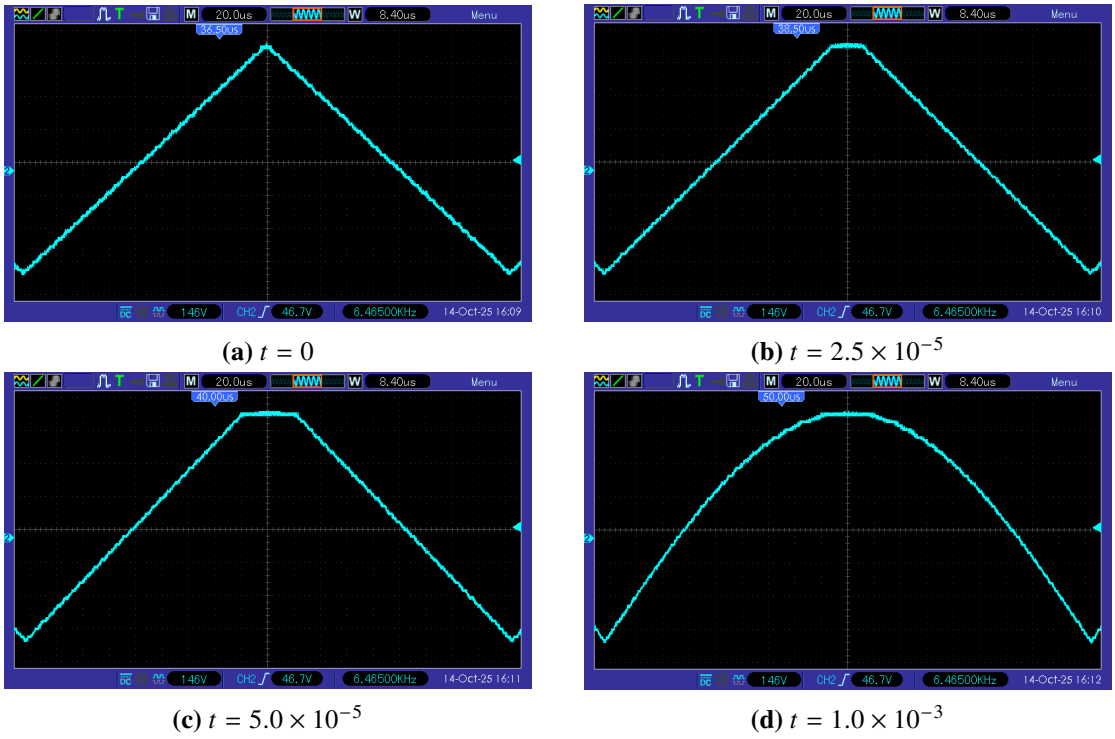


(a) $x = 0$ **(b)** $x = 20$

(c) $x = 40$ **(d)** $x = 60$

Fig. 10. Time-evolution of $\phi(x, t)$ for NTM displayed on DSO

The results clearly highlight the computational benefits and demonstrate the performance gains achieved by the proposed parallel computing algorithms, which exploit the GPU's computational capabilities.



(a) $t = 0$ **(b)** $t = 2.5 \times 10^{-5}$

(c) $t = 5.0 \times 10^{-5}$ **(d)** $t = 1.0 \times 10^{-3}$

Fig. 11. Spatial Evolution of $\phi(x,t)$ for NTM displayed on DSO

Table 2. Speed-up in execution time for evaluation of ϕ

	Execution Time (s)		
$\phi(x,t)$ evaluated for	Sequential (T_{SEQ})	GPU (T_{GPU})	Speed-up (T_{SEQ}/T_{GPU})
NDM	300.14	74.17	4.00
NTM	996.84	120.40	8.30

This speed-up is primarily due to the massive thread-level parallelism offered by the GPU, which allows simultaneous evaluation of spatial points and time steps in the solution using SoV method. By effectively leveraging CUDA features such as domain decomposition and memory hierarchy optimization, the proposed strategy minimizes data transfer overhead and maximizes throughput. Consequently, the reduction in execution time highlights not only the efficiency of the parallel algorithms but also their suitability for handling large-scale, real-time nuclear reactor simulations where timely results are critical.

The mean-squared error (MSE) is a measure of how close the computed values are to the reference values. Table 3 and Table 4 present the MSE values calculated for the neutron flux $\phi(x,t)$ obtained by comparing the proposed GPU-based CUDA implementation ($\phi_{Computed}$) to the CPU-based MATLAB implementation ($\phi_{Reference}$). The values are calculated for both NDM and NTM for all time instants at selected spatial locations and at all space locations for selected time instants. The MSE between the computed and reference values of $\phi(x,t)$ can be calculated in two ways,

(a) All t for a selected x

For a fixed spatial location x_k, the MSE is defined as:

$$\text{MSE}(x_k) = \frac{1}{N_t} \sum_{j=1}^{N_t} \left[\phi_{\text{Reference}}(x_k, t_j) - \phi_{\text{Computed}}(x_k, t_j)\right]^2, \tag{19}$$

Where, N_t is the total number of time steps.

(b) All x for a selected t

For a fixed time instant t_m, the MSE is given by:

$$\mathrm{MSE}(t_m) = \frac{1}{N_x} \sum_{i=1}^{N_x} \left[\phi_{\mathrm{Reference}}(x_i, t_m) - \phi_{\mathrm{Computed}}(x_i, t_m) \right]^2, \tag{20}$$

Where, N_x is the number of spatial grid points.

Table 3. Mean-squared error for $\phi(x,t)$ at selected spatial locations

Model	x (cm)	MSE
NDM	0	8.11×10^{-22}
	20	8.32×10^{-22}
	40	8.60×10^{-22}
	60	8.28×10^{-22}
NTM	0	4.56×10^{-32}
	20	4.60×10^{-32}
	40	3.23×10^{-32}
	60	3.83×10^{-32}

Table 4. Mean-squared error for $\phi(x,t)$ at selected time instants

Model	t (s)	MSE
NDM	0	7.54×10^{-22}
	2.5×10^{-5}	7.53×10^{-22}
	5.0×10^{-5}	7.68×10^{-22}
	1.0×10^{-3}	7.75×10^{-22}
NTM	0	2.03×10^{-32}
	2.5×10^{-5}	1.93×10^{-32}
	5.0×10^{-5}	1.97×10^{-32}
	1.0×10^{-3}	3.03×10^{-32}

These two formulations (19) and (20), quantify the deviation of computed results from reference data. From Table 3, it is evident that the deviation between the reference and computed results is minimal, on the order of 10^{-22} for NDM and 10^{-32} for NTM. This indicates that the parallel computation using the GPU produces results that are numerically consistent with the reference data. For the NDM, the MSE values remain fairly uniform across all spatial locations (x = 0, 20, 40, and 60 cm), showing a maximum of 8.60×10^{-22} and a minimum of 8.11×10^{-22} deviation. Such small deviation can be attributed to minor rounding differences due to the different precision handling mechanisms of the CPU and GPU arithmetic pipelines. For the NTM, the MSE values are even smaller, below 5.0×10^{-32}, confirming high numerical stability of the CUDA implementation, even with the added computational complexity from second-order time derivatives in the telegraph model. The slight variation among different spatial points is negligible, showing that the GPU maintains spatial consistency in flux evaluation.

Table 4 illustrates the MSE values at selected time instants. For the NDM, the MSE values range between 7.53×10^{-22} and 7.75×10^{-22} , showing minimal variation across time. This indicates that the GPU implementation preserves the exponential time-decay characteristics of the diffusion model with negligible temporal drift. For the NTM, the MSE remains consistently very low, from 1.93×10^{-33} to 3.03×10^{-32}, even at longer simulation times. This confirms that the proposed CUDA algorithm accurately reproduces the wave-like transient behaviour in the telegraph model without numerical instability or phase errors.

6. Discussion

The experimental results presented in this study clearly demonstrate the advantages of GPU-based parallelization for solving the diffusion model and the telegraph model. The embedded GPU platform NVIDIA Jetson Orin NX, provided sufficient computational resources to significantly reduce execution time while maintaining energy efficiency by using a compact system design. This demonstrates that nuclear reactor modelling can be effectively deployed on embedded GPU systems, achieving near real-time performance. The key enablers of this performance are the proposed algorithms implemented using the CUDA programming framework, which provides fine-grained control over the GPU hardware. The NDM involves exponential decay functions in time, and cosine terms in space, which are straightforward to parallelize but limited by the smaller number of arithmetic operations per iteration. The NTM, on the other hand, includes second-order time derivatives and additional hyperbolic or trigonometric evaluations, which significantly increase arithmetic intensity. This leads to better GPU utilization, resulting in higher overall speed-up. By exploiting thread-level parallelism, the large number of flux evaluations across spatial and temporal grids was distributed efficiently across hundreds of CUDA cores. This eliminated the bottleneck of serial computation and increased computational efficiency.

Memory management optimizations also played an essential role in achieving the reported speedups. Frequently used parameters such as eigen-coefficients and mode constants were stored in constant memory, which provides fast, cached access for all threads. This minimized global memory bandwidth usage and improved latency performance. Similarly, the use of shared memory within each thread block allowed cooperative data reuse, especially for spatial and temporal values accessed repeatedly by neighbouring threads. This reduced redundant global memory fetches and ensured higher arithmetic intensity per memory transaction. The separation of kernels for NDM and NTM models allowed algorithm-specific optimization. For instance, NTM requires handling wave-like terms involving both exponential and hyperbolic/oscillatory functions, which were efficiently managed by separating oscillatory and hyperbolic branches inside the kernel. Thread organization through `dim3` blocks and grids ensured memory coalescing, improving throughput further.

The low MSE values calculated validate that the proposed embedded GPU implementation reproduces the analytical SoV-based results with high fidelity while achieving significant computational speed-up. This establishes that the parallelization strategy, including memory optimization, kernel design, and thread-level computation, preserves numerical accuracy while providing superior performance. Overall, the results demonstrate that embedded GPUs provide a robust and practical platform for executing computationally intensive simulations efficiently. They strike a balance between performance, power consumption, and portability, making them particularly relevant for reactor monitoring and control systems where real-time response is crucial. This work measures the GPU kernel execution time. The DAC/DSO experiments demonstrate real-time signal generation but constitute an open-loop demonstration rather than a full HiL control experiment. The full end-to-end latency can be measured through an external control network (UDP/EtherCAT/ROS2). Such system-level latency depends on middleware and controller hardware.

The proposed approach satisfies the computational and timing requirements for HiL, but a complete closed-loop integration with a controller can be explored in a future study. The proposed GPU–SoV formulation scales efficiently for 1D reactor models and provides real-time performance on embedded hardware. This makes it attractive for digital-twin or control applications, and extending this approach to complex geometries or multi-group kinetics. Although the proposed GPU-SoV approach performs efficiently for NDM and NTM, the factors that can affect the implementation of this approach for complex geometries and multi-group kinetics include increased memory demands, degradation of accuracy, and floating-point performance limitations on embedded GPUs.

7. Conclusion

This work presented a GPU-accelerated implementation of nuclear reactor models, the neutron diffusion model and the neutron telegraph model, on the NVIDIA Jetson Orin NX platform equipped with an onboard Arm Cortex CPU and embedded GPU with Ampere architecture. These models play a pivotal role in the design and testing of controllers in HiL setup. By exploiting massive parallelism, optimized memory hierarchy, and efficient thread organization offered by CUDA, the proposed algorithms achieved substantial speed-up over traditional CPU-based implementations. The results demonstrate that even compact embedded GPUs can efficiently execute reactor-scale simulations with high accuracy, establishing their potential for real-time monitoring and control of nuclear reactors. The findings confirm that embedded GPUs provide a promising and energy-efficient alternative to large-scale high-performance computing systems for nuclear reactor simulations. Their compact size, low power consumption, and capability to deliver near real-time results make them highly suitable for deployment in safety-critical and field-based applications.

The possible extension of this work includes parallel computation and real-time implementation of multi-group neutron diffusion and telegraph formulations, and nodal models enabling a more realistic representation of reactor physics. Further optimization using mixed-precision arithmetic can be explored to enhance computational throughput while maintaining accuracy and energy efficiency. Additionally, integrating embedded GPU-based solvers into digital twin frameworks could enable continuous, real-time reactor monitoring and predictive safety assessments. The exploration of multi-GPU systems or distributed embedded GPU clusters presents another promising direction for achieving full-core, three-dimensional reactor modelling. Also, the fast implementation of nuclear reactor models on an embedded GPU platform demonstrated in this work can be used for the development of HiL systems for reactors.

Author Contribution: All authors contributed equally to the main contributor to this paper. All authors read and approved the final paper.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] G. Zhou and D. Tan, "Review of nuclear power plant control research: Neural network-based methods," *Annals of Nuclear Energy*, vol. 181, p. 109513, 2023, https://doi.org/10.1016/j.anucene.2022.109513.

[2] M. Khaleel *et al*., "Harnessing nuclear power for sustainable electricity generation and achieving zero emissions," *Energy Exploration & Exploitation*, vol. 43, no. 3, pp. 1126–1148, 2025, https://doi.org/10.1177/01445987251314504.

[3] J. C. Lee, *Nuclear Reactor Physics and Engineering*, John Wiley & Sons, 2020, https://books.google.co.id/books?id=u6o1EQAAQBAJ&hl=id&source=gbs_navlinks_s.

[4] J. J. Duderstadt and L. J. Hamilton, *Nuclear Reactor Analysis*, John Wiley and Sons, 1976, https://inis.iaea.org/records/fn4n6-kcn11.

[5] W. M. Stacey, *Nuclear Reactor Physics*, John Wiley & Sons, 2018, https://books.google.co.id/books?id=NzlJDwAAQBAJ&lr=&hl=id&source=gbs_navlinks_s.

[6] J. R. Lamarsh, *Introduction to Nuclear Reactor Theory*, Addison-Wesley Publishing Company, 1967, https://doi.org/10.1063/1.3034192.

[7] S. Glasstone and A. Sesonske, *Nuclear Reactor Engineering: Reactor Systems Engineering*, Springer Science & Business Media, 2012, https://books.google.co.id/books?id=lEbVBwAAQBAJ&hl=id&source=gbs_navlinks_s.

[8] A. F. Henry, "Continued development of nodal methods for nuclear reactor analysis," *Massachusetts Institute of Technology*, 1986, https://dspace.mit.edu/bitstream/handle/1721.1/29479/MIT-EL-86-002-19764322.pdf?sequence=1.

[9] A. Tiwari, T. Bhatt, and P. V. Surjagade, "Modelling and spatial control of 540 MWe pressurized heavy water reactor," *Transactions of the Indian National Academy of Engineering*, vol. 6, no. 3, pp. 731–753, 2021, https://doi.org/10.1007/s41403-021-00218-x.

[10] Y. Nagaya and K. Kobayashi, "Solution of 1-D multi-group time-dependent diffusion equations using the coupled reactors theory," *Annals of Nuclear Energy*, vol. 22, no. 7, pp. 421–440, 1995, https://doi.org/10.1016/0306-4549(94)00083-Q.

[11] A. E. Aboanber, "Spectral effects induced by the presence of a reflector for two-energy group two-point kinetic model of reflected reactors," *Progress in Nuclear Energy*, vol. 52, no. 2, pp. 197–205, 2010, https://doi.org/10.1016/j.pnucene.2009.06.004.

[12] A. A. Nahla, F. A. Al-Malki, and M. Rokaya, "Numerical techniques for the neutron diffusion equations in the nuclear reactors," *Advanced Studies in Theoretical Physics*, vol. 6, no. 14, pp. 649–664, 2012, https://www.academia.edu/download/83285445/nahlaASTP13-16-2012.pdf.

[13] A. A. Nahla, "Efficient computational system for transient neutron diffusion model via finite difference and theta methods," *Annals of Nuclear Energy*, vol. 89, pp. 28–37, 2016, https://doi.org/10.1016/j.anucene.2015.09.029.

[14] M. Shqair, A. El-Ajou, and M. Nairat, "Analytical solution for multi-energy groups of neutron diffusion equations by a residual power series method," *Mathematics*, vol. 7, no. 7, p. 633, 2019, https://doi.org/10.3390/math7070633.

[15] S. Corno, S. Dulla, P. Picca, and P. Ravetto, "Analytical approach to the neutron kinetics of the non-homogeneous reactor," *Progress in Nuclear Energy*, vol. 50, no. 8, pp. 847–865, 2008, https://doi.org/10.1016/j.pnucene.2008.02.001.

[16] C. Z. Petersen, B. E. Bodmann, M. T. Vilhena, and R. C. Barros, "Recursive solutions for multi-group neutron kinetics diffusion equations in homogeneous three-dimensional rectangular domains with time dependent perturbations," *Kerntechnik*, vol. 79, no. 6, pp. 494–499, 2014, https://doi.org/10.3139/124.110434.

[17] C. Z. Petersen, B. E. Bodmann, M. T. Vilhena, S. Dulla, and P. Ravetto, "On the exact solution for the multi-group kinetic neutron diffusion equation in a rectangle," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, 2011, https://inis.iaea.org/records/qe7bg-32s08/files/48022283.pdf?download=1.

[18] C. Z. Petersen, M. T. Vilhena, S. Dulla, and P. Ravetto, "An analytical solution of the point kinetics equations with time-variable reactivity by the decomposition method," *Progress in Nuclear Energy*, vol. 53, no. 8, pp. 1091–1094, 2011, https://doi.org/10.1016/j.pnucene.2011.01.001.

[19] R. Bru, D. Genstar, J. Marin, G. Verdú, J. Mas, and T. Manteuffel, "Iterative schemes for the neutron diffusion equation," *Computers & Mathematics with Applications*, vol. 44, no. 10–11, pp. 1307–1323, 2002, https://doi.org/10.1016/S0898-1221(02)00258-4.

[20] E. Machorro, "Discontinuous Galerkin finite element method applied to the 1-D spherical neutron transport equation," *Journal of Computational Physics*, vol. 223, no. 1, pp. 67–81, 2007, https://doi.org/10.1016/j.jcp.2006.08.020.

[21] L. Mei, "Multigrid algorithms for solving multigroup neutron transport equations," *Applied Mathematics and Computation*, vol. 179, no. 2, pp. 473–483, 2006, https://doi.org/10.1016/j.amc.2005.11.168.

[22] M. Yousefi, A. Zolfaghari, A. Minuchehr, and M. Abbassi, "An arbitrary geometry finite element method for the adjoint neutron transport equation," *Annals of Nuclear Energy*, vol. 110, pp. 511–525, 2017, https://doi.org/10.1016/j.anucene.2017.06.038.

[23] J. D. Logan, "Partial differential equations in the life sciences," *Applied Partial Differential Equations*, pp. 172–196, 2004, https://doi.org/10.1007/978-1-4419-8879-9_5.

[24] S. J. Farlow, *Partial Differential Equations for Scientists and Engineers*, Courier Corporation, 1993, https://books.google.co.id/books?id=uRjhDAAAQBAJ&hl=id&source=gbs_navlinks_s.

[25] V. Ardourel and J. Jebeile, "On the presumed superiority of analytical solutions over numerical methods," *European Journal for Philosophy of Science*, vol. 7, pp. 201–220, 2017, https://doi.org/10.1007/s13194-016-0152-2.

[26] F. Oliveira, J. Fernandes, B. Bodmann, and M. Vilhena, "On an analytical solution for the two energy group neutron space-kinetic equation in heterogeneous cylindrical geometry," *Annals of Nuclear Energy*, vol. 133, pp. 216–220, 2019, https://doi.org/10.1016/j.anucene.2019.05.018.

[27] F. Oliveira, B. Bodmann, M. Vilhena, and F. Carvalho, "On an analytical formulation for the mono-energetic neutron space-kinetic equation in full cylinder symmetry," *Annals of Nuclear Energy*, vol. 99, pp. 253–257, 2017, https://doi.org/10.1016/j.anucene.2016.08.032.

[28] V. A. Vyawahare and P. S. V. Nataraj, *Fractional-Order Modelling of Nuclear Reactor: From Subdiffusive Neutron Transport to Control-Oriented Models: A Systematic Approach*, Springer, 2018, https://books.google.co.id/books?id=MFNKDwAAQBAJ&hl=id&source=gbs_navlinks_s.

[29] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," in *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008, https://doi.org/10.1109/JPROC.2008.917757.

[30] D. Zhao *et al*., "Sustainable supercomputing for AI: GPU power capping at HPC scale," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, USA, pp. 588–596, 2023, https://doi.org/10.1145/3620678.3624793.

[31] W. W. Hwu, *GPU Computing Gems Jade Edition*, Elsevier, 2011, https://books.google.co.id/books?id=MFNKDwAAQBAJ&hl=id&source=gbs_navlinks_s.

[32] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of CUDA and OpenCL," in *2011 International Conference on Parallel Processing*, pp. 216–225, 2011, https://doi.org/10.1109/ICPP.2011.45.

[33] Y. Wang, W. Chen, and V. Dinavahi, "Detailed transient modeling and FPGA-based real-time digital-twin development for sodium-cooled fast reactor," *Progress in Nuclear Energy*, vol. 189, p. 105890, 2025, https://doi.org/10.1016/j.pnucene.2025.105890.

[34] A. Messai, I. Abdellani, and A. Mellit, "FPGA-based real-time implementation of a digital reactivity-meter," *Progress in Nuclear Energy*, vol. 150, p. 104313, 2022, https://doi.org/10.1016/j.pnucene.2022.104313.

[35] V. -T. Vo *et al*., "A digital controller for reactivity monitoring and power control," *Science and Technology of Nuclear Installations*, vol. 2023, no. 1, p. 2839654, 2023, https://doi.org/10.1155/2023/2839654.

[36] D. J. Rankin and J. Jiang, "A hardware-in-the-loop simulation platform for the verification and validation of safety control systems," *IEEE Transactions on Nuclear Science*, vol. 58, no. 2, pp. 468–478, 2011, https://doi.org/10.1109/TNS.2010.2103325.

[37] P. Liu, C. Bi, Y. Yu, M. Lin, X. Song, G. Wu, and Z. Su, "Development of a hardware-in-the-loop simulation platform for NPP main control systems," *Matec Web of Conferences*, vol. 128, p. 02012, 2017, https://doi.org/10.1051/matecconf/201712802012.

[38] S. Sood, A. Malik, and P. Roop, "Robust hardware-software co-simulation framework for design and validation of hybrid systems," in *2022 20th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pp. 1–11, 2022, https://doi.org/10.1109/MEMOCODE57689.2022.9954590.

[39] V. Lakshminarayanan, C. Patabandi, O. Nayak, and B. Lopez, "HiL validation of power plant controller model," in *2022 North American Power Symposium (NAPS)*, pp. 1–6, 2022, https://doi.org/10.1109/NAPS56150.2022.10012177.

[40] D. P. Acharya, N. Hannon, S. Choudhury, N. Nayak, and A. Satpathy, "Design and hardware-in-loop testing of an intelligent controller for power quality improvement in a complex micro grid," *Energy Reports*, vol. 9, pp. 4135–4156, 2023, https://doi.org/10.1016/j.egyr.2023.03.032.

[41] D. Gurudev, M. Shaikh, P. Shah, R. Sekhar, and D. Nandan, "Role of hardware-in-the-loop testing in model-based design," in *2023 2nd International Conference on Futuristic Technologies*, pp. 1–6, 2023, https://doi.org/10.1109/INCOFT60753.2023.10425196.

[42] S. G. Karad *et al*., "Optimal design of fractional order vector controller using hardware-in-loop (HiL) and Opal RT for wind energy system," in *IEEE Access*, vol. 12, pp. 35033–35047, 2024, https://doi.org/10.1109/ACCESS.2024.3357504.

[43] Y. C. Keluskar, N. G. Singhaniya, V. A. Vyawahare, C. S. Jage, P Patil, G. E. -Paredes, "Solution of nonlinear fractional-order models of nuclear reactor with parallel computing: Implementation on GPU platform," *Annals of Nuclear Energy*, vol. 195, p. 110134, 2024, https://doi.org/10.1016/j.anucene.2023.110134.

[44] N. Choi, K. M. Kim, and H. G. Joo, "Optimization of neutron tracking algorithms for GPU-based continuous energy Monte Carlo calculation," *Annals of Nuclear Energy*, vol. 162, p. 108508, 2021, https://doi.org/10.1016/j.anucene.2021.108508.

[45] A. A. de Moura Meneses, L. M. Araujo, and R. Schirru, "A GPU-accelerated linear system solution for the Galerkin finite element method applied to neutron diffusion equation," *Nuclear Engineering and Design*, vol. 421, p. 113103, 2024, https://doi.org/10.1016/j.nucengdes.2024.113103.

[46] A. Zhang, M. Dai, M. Cheng, J. Wu, and J. Chen, "Development of a GPU based three-dimensional neutron transport code," *Annals of Nuclear Energy*, vol. 174, p. 109156, 2022, https://doi.org/10.1016/j.anucene.2022.109156.

[47] T. R. Phillips, C. E. Heaney, B. Chen, A. G. Buchan, and C. C. Pain, "Solving the discretised neutron diffusion equations using neural networks," *International Journal for Numerical Methods in Engineering*, vol. 124, no. 21, pp. 4659–4686, 2023, https://doi.org/10.1002/nme.7321.

[48] K. S. Smith, "Assembly homogenization techniques for light water reactor analysis," *Progress in Nuclear Energy*, vol. 17, no. 3, pp. 303–335, 1986, https://doi.org/10.1016/0149-1970(86)90035-1.

[49] T. Mullikin, "Estimates of critical dimensions of spherical and slab reactors," *Journal of Mathematical Analysis and Applications*, vol. 5, no. 2, pp. 184–199, 1962, https://www.osti.gov/biblio/4810016.

[50] R. Alcouffe, E. Larsen, W. Miller Jr, and B. Wienke, "Computational efficiency of numerical methods for the multigroup, discrete-ordinates neutron transport equations: The slab geometry case," *Nuclear Science and Engineering*, vol. 71, no. 2, pp. 111–127, 1979, https://doi.org/10.13182/NSE71-111.

[51] A. Compte and R. Metzler, "The generalized Cattaneo equation for the description of anomalous transport processes," *Journal of Physics A: Mathematical and General*, vol. 30, no. 21, p. 7277, 1997, https://doi.org/10.1088/0305-4470/30/21/006.

[52] B. Vick and M. N. Özisik, "Growth and decay of a thermal pulse predicted by the hyperbolic heat conduction equation," *ASME Journal of Heat Transfer*, vol. 105, no. 4, pp. 902–907, 1983, https://doi.org/10.1115/1.3245680.

[53] M. N. Özisik and D. Y. Tzou, "On the wave theory in heat conduction," *ASME Journal of Heat Transfer*, vol. 116, no. 3, pp. 526–535, 1994, https://doi.org/10.1115/1.2910903.

[54] X. Gong, R. Ubal and D. Kaeli, "Multi2Sim Kepler: A detailed architectural GPU simulator," *2017 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 269-278, 2017, https://doi.org/10.1109/ISPASS.2017.7975298.

[55] L. S. Karumbunathan, "NVIDIA Jetson AGX Orin Series: A Giant Leap Forward for Robotics and Edge AI Applications," *Technical Brief*, 2022, https://www.nvidia.cn/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf.

[56] T. Fukagai *et al*., "Speed-up of object detection neural network with GPU," in *2018 25th IEEE International Conference on Image Processing*, pp. 301–305, 2018, https://doi.org/10.1109/ICIP.2018.8451814.

[57] S. W. Ali *et al*., "On the necessity of real-time principles in GPU-driven autonomous robots," in *2025 IEEE International Conference on Robotics and Automation*, pp. 8086–8092, 2025, https://doi.org/10.1109/ICRA55743.2025.11128627.

[58] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU–Past, present and future," *Medical Image Analysis*, vol. 17, no. 8, pp. 1073–1094, 2013, https://doi.org/10.1016/j.media.2013.05.008.

[59] I. M. -Salinas *et al*., "Evaluating and accelerating vision transformers on GPU-based embedded edge AI systems," *The Journal of Supercomputing*, vol. 81, no. 1, p. 349, 2025, https://doi.org/10.1007/s11227-024-06807-1.

[60] M. A. Farooq, W. Shariff, and P. Corcoran, "Evaluation of thermal imaging on embedded GPU platforms for application in vehicular assistance systems," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 2, pp. 1130–1144, 2023, https://doi.org/10.1109/TIV.2022.3158094.

[61] D. Jaiswal and P. Kumar, "A comparative study on SoC embedded low power GPUs for real-time edge-based automated traffic surveillance," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 10, p. e6736, 2022, https://doi.org/10.1002/cpe.6736.

[62] F. Ya¸sa, F. Anli, and S. Güngör, "Eigenvalue spectrum with chebyshev polynomial approximation of the transport equation in slab geometry," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 97, no. 1, pp. 51–57, 2006, https://doi.org/10.1016/j.jqsrt.2004.12.017.