CPU-GPU Concurrent Computing Algorithm for Particle Transport Using Discontinuous Finite Element Discrete Ordinates on Unstructured Grids

Authors: Kong, Dr. Boran, Dai, Dr. Tao, Dr. Longfei Xu, Li, Dr. Baiwen, Dai, Dr. Ni, Xiao, Dr. Bowen, Dr. Longfei Xu

Date: 2025-04-08T17:48:32+00:00

Abstract

Discontinuous finite element method (DFEM) based discrete ordinates (SN) for solving particle transport is a heavy burden that costs plenty of calculation time in multi-physics simulation. The emergence of the graphic processing unit (GPU) has triggered a change in computing capabilities, which provides a new way for DFEM SN calculation. However, the effect of the general GPU acceleration algorithm for particle transport with unstructured grids is limited due to the dependence of the SN spatial wavefront, and the computing capability of GPU is not fully utilized. This paper proposes a CPU-GPU concurrent computing algorithm, which is carried out as a large-scale linear system with a block Jacobi parallel strategy. In the algorithm, the fission and scattering source terms are calculated on the GPU, and the inflow terms are calculated on the CPU simultaneously. Moreover, the coarse-grained domain decomposition parallelism and fine-grained angular parallelism are adopted, where the data transmission and computation are performed simultaneously. This new algorithm not only takes full advantage of the huge number of threads on the GPU but also has no limits to the total GPU utilization. Numerical results of typical neutron transport benchmarks show that the CPU-GPU concurrent computing algorithm achieves a 24-60 times acceleration effect than the CPU algorithm and 6-11 times acceleration than the general GPU algorithm.

Full Text

Preamble

CPU-GPU Concurrent Computing Algorithm for Particle Transport Using Discontinuous Finite Element Discrete Ordinates with Un-

structured Grids

Boran Kong[1], Tao Dai[1], Longfei Xu[1], *Baiwen Li*[1], *Ni Dai*[1], *Bowen Xiao*[1]
[1] *Institute of Applied Physics and Computational Mathematics, Beijing 100094, China*
xu_{longfei}@iapcm.ac.cn*

Abstract: The discontinuous finite element method (DFEM) based discrete ordinates (SN) method for solving particle transport problems imposes a heavy computational burden, consuming substantial calculation time in multi-physics simulations. The emergence of graphics processing units (GPUs) has transformed computing capabilities, offering a new approach for DFEM-SN calculations. However, conventional GPU acceleration algorithms for particle transport with unstructured grids achieve limited effectiveness due to dependencies in the SN spatial wavefront, leaving GPU computing power underutilized. This paper proposes a CPU-GPU concurrent computing algorithm formulated as a large-scale linear system using a block Jacobi parallel strategy. In this algorithm, fission and scattering source terms are computed on the GPU while inflow terms are calculated on the CPU through coarse-grained domain decomposition. Furthermore, decomposition parallelism and fine-grained angular parallelism are employed, enabling simultaneous data transmission and computation. This novel algorithm not only leverages the massive number of threads on GPUs but also eliminates constraints on total GPU utilization. Numerical results from typical neutron transport benchmarks demonstrate that the CPU-GPU concurrent computing algorithm achieves 24–60× speedup over CPU-only algorithms and 6–11× speedup over general GPU algorithms.

Keywords: GPU; Concurrent computing; Particle transport; Discrete ordinates; Unstructured grid; Discontinuous finite element

1. Introduction

The particle transport equation plays a crucial role in modeling numerous scientific domains, with significant applications in nuclear reactor analysis, radiation physics, high-energy physics, and related fields [1-3]. This equation is multi-dimensional, involving spatial, angular, energy, and temporal variables, and can consume 50%–80% of total computational time in Department of Energy (DOE) systems [1,4]. The discrete ordinates (SN) method is widely employed for solving the transport equation by selecting representative discrete angles for angular discretization [5]. Meanwhile, to accommodate increasingly complex geometric designs and large deformation problems, SN methods often adopt the discontinuous finite element method (DFEM) for spatial discretization [6]. However, DFEM-SN for particle transport problems demands substantial memory and computational time [7].

With the advancement of high-performance computing (HPC), extensive research has been conducted on DFEM-SN parallelization for unstructured grids. Plimpton proposed an asynchronous algorithm based on grid partitioning and

cycle detection [8]. Pautz described a low-complexity list-ordering heuristic for determining the sweeping sequence [9,10]. Mo developed a parallel flux sweep algorithm based on domain decomposition [11]. The essence of these algorithms lies in converting the element sweep order into a directed acyclic graph (DAG) [10]. Numerous efforts have been made to handle cyclic dependencies during transport sweeps [12-14] and improve parallel efficiency on central processing units (CPUs) [13-15]. On the other hand, heterogeneous CPU-GPU systems have developed rapidly [16,17], providing new methods for high-fidelity DFEM-SN transport calculations.

Gong et al. demonstrated particle transport simulations on unstructured grids using GPUs, achieving 11.03× speedup on an M2050 GPU compared to an Intel X5355 processor [1,18]. However, the data dependency of wavefronts [19] limited parallelism and prevented full utilization of GPU computing capability.

Modern HPC architectures incorporate many-core CPUs and GPUs, enabling simultaneous execution on both processors [17,20-21]. To overcome the inherent parallelism limitations of wavefront sweeping and adapt to contemporary computing architectures, this paper proposes a CPU-GPU concurrent computing DFEM-SN algorithm. The algorithm is designed in the form of a large-scale linear system using a hybrid Message Passing Interface (MPI) [22] programming model. Domain and Compute Unified Device Architecture (CUDA) decomposition [23] are implemented as coarse-grained parallelism, with each subdomain assigned to a specific CPU processor. A block Jacobi parallel strategy [24] is then applied: source terms are calculated on GPUs while inflow terms are simultaneously computed on CPUs through coarse-grained domain decomposition. Angular parallelism is implemented as fine-grained parallelism based on GPU asynchronous streaming operations and Multi-Process Service (MPS) [22]. Compared with traditional DAG-based CPU algorithms and general GPU acceleration algorithms, this new CPU-GPU concurrent algorithm achieves significant speedup in typical pressurized water reactor (PWR) [25] neutronics benchmarks.

The remainder of this paper is organized as follows. Section 2 presents the methodologies, including the SN transport equation, DFEM discretization, and a brief comparison between block Jacobi and Gauss-Seidel parallel strategies. Section 3 describes the detailed implementation of the CPU-GPU concurrent computing algorithm. Section 4 presents performance evaluations on typical PWR benchmarks and related discussions. Section 5 summarizes conclusions.

2.1 SN Transport Equation

Particle transport through various media is described by the Boltzmann equation, derived from particle balance conditions [26,27]. Using neutrons as an example (the steady-state Boltzmann equation has a similar form for other particles like photons and electrons):

$$\cdot\nabla\psi(\mathbf{r},E,\ )+\Sigma_t(\mathbf{r},E)\psi(\mathbf{r},E,\ )=\int_0^\infty dE'\int_{4\pi}d\ '\Sigma_s(\mathbf{r},E'\to E,\ '\cdot\ )\psi(\mathbf{r},E',\ ')+\frac{\chi(\mathbf{r},E)}{k_{\text{eff}}}\int_0^\infty dE'\int_{4\pi}d\ '\nu\Sigma_f(\mathbf{r},I$$

where $\psi(\mathbf{r},E,\ )$ represents the angular flux as a function of position $\mathbf{r}$, energy E, and angular direction . The scalar flux $\phi(\mathbf{r},E)$ is obtained through angular integration of the angular flux. The left-hand side (LHS) terms represent streaming and collision removal, respectively, while the right-hand side (RHS) represents neutron gain from scattering and fission. Here, Σ_t, Σ_s, and Σ_f denote the total, scattering, and fission cross-sections, respectively; χ is the fission spectrum; k_{eff} is the effective eigenvalue; and ν is the number of fission neutrons per fission. For numerical solution using deterministic methods, the basic problem is discretized in space, angle, and energy [18,28].

In this study, energy variables are discretized using the multi-group method [29], and the anisotropic scattering term is simplified through isotropic scattering approximation. Angle variables are discretized by the SN method. The multi-group SN transport equation under 2D Cartesian coordinates becomes:

$$_m\cdot\nabla\psi_{g,m}(\mathbf{r})+\Sigma_{t,g}(\mathbf{r})\psi_{g,m}(\mathbf{r})=\sum_{g'=1}^{G}\Sigma_{s,g'\to g}(\mathbf{r})\phi_{g'}(\mathbf{r})+\frac{\chi_g(\mathbf{r})}{k_{\text{eff}}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(\mathbf{r})\phi_{g'}(\mathbf{r})$$

where subscript m represents the angular index and g represents the energy group index. The scalar flux is calculated from the angular flux as shown in Eq. (3).

2.2 DFEM Discretization

Spatial variables in Eq. (2) are discretized using DFEM, which is stable and accurate for solving first-order equations [30]. DFEM integrates Eq. (2) over elements with weighting functions $w(\mathbf{r})$:

$$\int_{V_e}w(\mathbf{r})\ _m\cdot\nabla\psi_{g,m}(\mathbf{r})dV+\int_{V_e}w(\mathbf{r})\Sigma_{t,g}(\mathbf{r})\psi_{g,m}(\mathbf{r})dV=\int_{V_e}w(\mathbf{r})Q_{g,m}(\mathbf{r})dV$$

The total source term on the RHS contains scattering and fission source terms. Applying the divergence theorem to the streaming term, Eq. (4) can be written as:

$$\int_{V_e}{}_m\cdot\nabla w(\mathbf{r})\psi_{g,m}(\mathbf{r})dV+\int_{V_e}w(\mathbf{r})\Sigma_{t,g}(\mathbf{r})\psi_{g,m}(\mathbf{r})dV+\int_{\partial V_e}w(\mathbf{r})\ _m\cdot\mathbf{n}\psi^+_{g,m}(\mathbf{r})dS=\int_{V_e}w(\mathbf{r})Q_{g,m}(\mathbf{r})dV+\int_{\partial V_e}w(\mathbf{r})\ ,$$

Unlike continuous FEM, DFEM permits discontinuity in the surface integration term, as shown in the first term of Eq. (5). The superscript + indicates the location outside the element surface, while − indicates the location inside. DFEM here specifically refers to the linear discontinuous Galerkin finite element method, where weight functions remain identical to trial functions [25]. The angular flux and source term are linearly expanded using Nth-order basis functions:

$$\psi_{g,m}(\mathbf{r}) = \sum_{i=1}^{N} \psi_{g,m}^{i} b_i(\mathbf{r}), \quad Q_{g,m}(\mathbf{r}) = \sum_{i=1}^{N} Q_{g,m}^{i} b_i(\mathbf{r})$$

Substituting Eq. (7) into Eq. (5), the matrix form of the DFEM-SN equation becomes:

$$\mathbf{A}_{g,m}^{e} \psi_{g,m}^{e} = \mathbf{S}_{g,m}^{e} + \sum_{\text{neighbor}} \mathbf{B}_{g,m}^{e,\text{neighbor}} \psi_{g,m}^{\text{neighbor}}$$

where

$$\mathbf{A}_{g,m}^{e} = \int_{V_e} {}_{m} \cdot \nabla b_i(\mathbf{r}) b_j(\mathbf{r}) dV + \int_{V_e} \Sigma_{t,g}(\mathbf{r}) b_i(\mathbf{r}) b_j(\mathbf{r}) dV + \int_{\partial V_e^{+}} {}_{m} \cdot \mathbf{n} b_i(\mathbf{r}) b_j(\mathbf{r}) dS$$

$$\mathbf{B}_{g,m}^{e,\text{neighbor}} = \int_{\partial V_e^{-}} {}_{m} \cdot \mathbf{n} b_i(\mathbf{r}) b_j(\mathbf{r}) dS$$

$$\mathbf{S}_{g,m}^{e} = \int_{V_e} Q_{g,m}(\mathbf{r}) b_i(\mathbf{r}) dV$$

The outflow term appears on the LHS, while the inflow term appears on the RHS and is treated as a known variable during sweeping.

2.3 Parallel Strategies

The DFEM-SN equation in Eq. (8) is typically solved iteratively through loops over energy, space, and angle, as shown in Algorithm 1. For neutron transport calculations, an inner-outer iterative strategy is widely adopted to accelerate convergence. During inner iterations, the scattering source term is updated while the fission source term and eigenvalue remain constant. The effective eigenvalue and fission source term are updated only during outer iterations [5,28]. The energy loop typically proceeds from high to low energy groups, facilitating updates to down-scattering terms [23,29]. The angle loop has no specific sequence requirements under 2D Cartesian coordinates. For a given direction, the sweeping sequence on unstructured grids can be determined by a DAG, as

illustrated in [Figure 1: see original paper]. Each node represents a triangular grid element, and arrows between nodes denote dependency relationships. The DAG defines cells by mapping each grid with each angular direction, with a cell' s in-degree defined as the sum of its inflow counts [8-11]. Detailed DAG theory can be found in Refs. [8-11].

Algorithm 1: DFEM-SN Iteration Calculation

```
While (Not converged) Outer iteration {
    For (g=1:G) energy loop {
        While (Not converged || Reach specific number of inner iterations) Inner iteration {
            Update scattering source term
            For (m=1:M) angle loop {
                Sweep over whole space domain
            }
            End For (m=1:M)
        }
        End while (Inner iteration)
    }
    End For (g=1:G)
    Update fission source term and eigenvalue
}
End While (Outer iteration)
```

[Figure 1: see original paper] An illustration of DAG and domain decomposition

In [Figure 1: see original paper], the entire computational domain is partitioned into several subdomains assigned to corresponding processors. Taking processor 2 as an example, it receives incoming information from processor 1 and sends outgoing information to processor 3. Boundary node calculations depend on messages from other processors. Two parallel strategies are widely adopted: Gauss-Seidel and block Jacobi. In the Gauss-Seidel strategy, after receiving incoming messages from other processors, relevant boundary cells decrease their in-degree. When the in-degree reaches zero, the boundary cell becomes calculable, enabling subsequent computations. In the block Jacobi strategy, boundary cells use delayed boundary information from other processors, with the in-degree from other processors set to zero.

A key difference between the two strategies is that block Jacobi requires only unified communication after completing the entire sweep, whereas Gauss-Seidel requires continuous message passing during sweeping. Block Jacobi uses delayed boundary information, reducing communication and waiting time at the cost of increased outer iterations [24].

To improve block Jacobi convergence, energy group looping is widely applied in practice. A cell' s in-degree is independent of energy index. To enhance parallel efficiency, the Gauss-Seidel strategy computes cells across all energy groups at each sweeping step, abandoning the energy group loop. Consequently, the iterative calculations differ between the two strategies: block Jacobi uses Algorithm

1, while Gauss-Seidel uses Algorithm 2. After an inner iteration, scattering terms for all energy groups are unified and updated in the Gauss-Seidel strategy. Compared with Algorithm 1, Algorithm 2 exhibits more significant lag in down-scattering terms as the number of energy groups increases. Additionally, for block Jacobi, the sweeping sequence remains identical for each inner iteration, allowing pre-calculation and storage of the sweeping sequence to reduce DAG determination overhead. In contrast, Gauss-Seidel must determine the sweeping sequence online for each inner iteration, representing a non-trivial burden. Both block Jacobi and Gauss-Seidel parallel strategies have distinct advantages and are used in neutron transport calculations. Many improvements have been made to enhance parallel efficiency with significant success [31,32], though these are beyond this paper' s scope.

Algorithm 2: Gauss-Seidel Iteration Calculation

```
While (Not converged) Outer iteration {
    While (Not converged || Reach specific number of inner iterations) Inner iteration {
        Update scattering source term of all energy groups
        Calculate cells for all energy groups
    }
    End while (Inner iteration)
    Update fission source term and eigenvalue
}
End While (Outer iteration)
```

3.1 Framework of the DFEM-SN Code SNIPER

To implement CPU-GPU concurrent computing and compare different GPU algorithms, a DFEM-SN code named SNIPER [33] was developed. SNIPER is programmed in C++ and utilizes a hybrid MPI and CUDA programming model. The framework of the steady-state SNIPER code is shown in [Figure 2: see original paper]. The code employs the mature graph partitioning software METIS from Karypis Lab for domain decomposition [34]. Control conditions, material parameters, and iteration indices are specified in input and material cards. Flux distributions, geometry, and material information are verified and displayed in output cards. The code is organized into several classes: transport information, mesh information, sweeping, integration, parallelization, and calculation. The transport information class sets basic parameters such as cross-sections and quadrature sets according to input cards. The mesh information class reads unstructured meshes from msh-format mesh files and interfaces with METIS for domain decomposition. The parallel class accumulates parallelization information including computation time, communication time, and waiting time. Calculation of Eq. (8) is primarily implemented in the calculation class, while integration terms in Eq. (8) are computed in the integration class. SNIPER supports both Gauss-Seidel and block Jacobi parallel strategies. For block Jacobi, the sweeping sequence is pre-calculated and stored, while for Gauss-Seidel, it is determined online via DAG.

[Figure 2: see original paper] Framework of DFEM-SN code SNIPER

3.2 General GPU Acceleration Algorithm

The emergence of GPUs has transformed computing capabilities, rooted in their parallel architectures that enable simultaneous handling of numerous tasks [35]. GPUs excel at massive Single Instruction Multiple Data (SIMD) processing, making them ideal for repetitive, parallelizable tasks. Additionally, GPUs feature high-bandwidth memory hierarchies crucial for data-intensive computations [36,37]. In CPUs, chip cache and control logic occupy relatively large areas, limiting core count. Conversely, GPUs contain many more streaming multiprocessors (SMs). In the CUDA programming model, coarse-grained parallelism is used for task-level macro-operations among SMs, while fine-grained parallelism handles specific data operations within a single SM, delivering high parallel performance [38]. In summary, CPUs with complex control logic and large caches reduce latency and are suitable for general-purpose computing, especially complex logical operations. GPUs, benefiting from massive thread counts, are applied to large-scale parallel tasks with high computational density and simple logical branches.

For realistic problems, solving the DFEM-SN equation (8) represents a massive computational burden requiring over 10^{13} degrees of freedom [1,4]. GPUs provide a new approach for solving Eq. (8), primarily for accelerating large-scale computations. Flowcharts of GPU acceleration for unstructured grid neutron transport are shown in [Figure 3: see original paper] and [Figure 4: see original paper]. [Figure 3: see original paper] illustrates the Gauss-Seidel parallel strategy in the MPI and CUDA programming model. To prevent iteration degradation, pipeline SN sweeping is utilized, requiring frequent CPU-GPU communication. The GPU acceleration algorithm for the block Jacobi parallel strategy is shown in [Figure 4: see original paper], where updated information is uniformly communicated after sweeping over all unknowns.

[Figure 3: see original paper] Flowchart of GPU acceleration algorithm in Gauss-Seidel parallel strategy
[Figure 4: see original paper] Flowchart of GPU acceleration algorithm in block Jacobi parallel strategy

Four kernel functions are employed on the GPU: scattering source update, inflow term calculation, cell calculation, and fission source term update. The scattering source and fission source updates in Eq. (4) are suitable for parallel computation and easy to implement on GPUs. The inflow term calculation in Eq. (9) uses neighbor angular flux and requires branch operations including CPU ownership judgment and reflective boundary handling. For DFEM with iso-parametric elements [39], neighbor side ownership must also be determined. Such extensive branching reduces GPU efficiency. [Figure 5: see original paper] shows the number of calculable elements at each sweeping step using the block Jacobi parallel strategy in a typical PWR neutronics benchmark. The GPU thread

count for kernels 2 and 3 is determined by the number of calculable elements, with a maximum of 914 and minimum of only 4. The number of thread blocks changes dramatically across sweeping steps, meaning GPU computing capability is not fully utilized. Wavefront dependence in the sweeping sequence limits parallelism and represents the bottleneck of GPU acceleration [1,18].

[Figure 5: see original paper] The number of calculable elements in each sweeping step

3.3 GPU Acceleration Algorithm in Large-Scale Linear System Form

To fully utilize GPU computing capability, a new GPU acceleration algorithm is necessary. When solving Eq. (8) in pipeline sweeping mode, insufficient GPU resource utilization is a fundamental problem. Moreover, for small-scale matrix computations, GPU advantages over CPU are not obvious [35-37,40]. Therefore, a new GPU acceleration algorithm is designed in large-scale linear system form.

The DFEM-SN equation (8) can be expressed in both linear system and sweeping forms, briefly illustrated in [Figure 6: see original paper]. Equation (8) for all elements in the entire domain is unified into a large-scale linear system $\mathbf{Ax} = \mathbf{Q}$, where $\mathbf{x}_1$ represents variables of the first grid, and so on. Note that points in $\mathbf{x}$ are not single variables but sets of all unknowns for a grid. Based on DAG, the first and second grids can be calculated first. Due to grid dependencies, the third and fourth grids are calculated sequentially. The sweeping form can be understood as a physical method for solving the large-scale linear system, theoretically an optimization approach with clear physical meaning. The linear system is sparse and can be solved by Krylov subspace methods such as the Generalized Minimal Residual (GMRES) algorithm [41,42]. While GMRES loses physical meaning and is not superior to sweeping under CPU architecture, it conceals element dependencies in the sweeping sequence, enabling full utilization of massive GPU threads and offering a potential solution for Eq. (8) on GPU architecture.

Based on this analysis, a new GPU acceleration algorithm is proposed in [Figure 7: see original paper]. The entire calculation subdomain for each discretized angle is uniformly written into a sparse linear system. Boundary inflow terms from other CPUs are set as known variables and added to the RHS of the linear system, uniformly updated and transmitted to the GPU after each outer iteration. Consequently, only the block Jacobi parallel strategy can be adopted in this new GPU acceleration algorithm. The sparse matrix $\mathbf{A}$ is assembled on the CPU in Compressed Sparse Row (CSR) format and transferred to GPUs during preparation. Traditional GMRES increases subspace dimension and data storage during computation, which are difficult operations on GPUs. Therefore, the restarted GMRES algorithm with fixed subspace dimension [42] is utilized to solve the sparse linear system. Unlike general GPU acceleration algorithms, calculable element inflow terms and element dependencies are concealed in the

sparse linear system, and restarted GMRES calculations for all discretized angles are performed on GPUs during each inner iteration.

[Figure 6: see original paper] Transport calculation in linear system form and sweeping form
[Figure 7: see original paper] GPU acceleration algorithm in large-scale linear system form

The restarted GMRES algorithm is listed in Algorithm 3. Three main aspects limit GPU efficiency: (1) 2-norm calculations (step 2) require global information reduction implemented via on-chip shared memory, resulting in low GPU utilization; (2) step 3(3) checks for restarted GMRES interruption and involves branch operations inefficient on GPUs; (3) step 5 contains Givens rotation operations with minimal parallelism, preventing full GPU resource utilization. Despite utilizing more threads, these limitations restrict total GPU utilization in the new large-scale linear system GPU acceleration algorithm.

Algorithm 3: Restarted GMRES Algorithm for $\mathbf{Ax} = \mathbf{Q}$ Sparse Linear System

1. Initialize restart parameter m, initial guess $\mathbf{x}_0$ (flux from last iteration)
2. Calculate $\mathbf{r}_0 = \mathbf{Q} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, $\mathbf{v}_1 = \mathbf{r}_0/\beta$
3. For $j = 1 : m$ { $\mathbf{w} = \mathbf{A}\mathbf{v}_j$ For $i = 1 : j$ { $h_{i,j} = (\mathbf{w}, \mathbf{v}_i)$ $\mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$ } End For $h_{j+1,j} = \|\mathbf{w}\|_2$ $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$ If $h_{j+1,j}$ is small, break to step 4 }
4. Solve least squares problem $\arg\min \|\beta\mathbf{e}_1 - \bar{H}_j\mathbf{y}\|_2$ by Givens rotation, $\mathbf{x}_m = \mathbf{x}_0 + V_j\mathbf{y}$
5. Calculate $\|\mathbf{r}_m\|_2 = \|\mathbf{Q} - \mathbf{A}\mathbf{x}_m\|_2$. If $\|\mathbf{r}_m\|_2 <$ tol, solution found; else set $\mathbf{x}_0 = \mathbf{x}_m$ and return to step 2

3.4 CPU-GPU Concurrent Computing Algorithm

As discussed in Section 3.3, GPU acceleration in large-scale linear system form enables massive GPU thread utilization, but restarted GMRES for solving linear systems limits total GPU utilization. Meanwhile, modern HPC architectures contain many-core CPUs and GPUs, enabling simultaneous execution on both processors. Inspired by these considerations, a CPU-GPU concurrent computing algorithm is proposed.

Similar to Section 3.3, the large-scale linear system form of transport calculation is retained. Non-diagonal block elements in matrix $\mathbf{A}$ are rearranged into the RHS of the linear system. Instead of Krylov subspace methods, a block-Jacobi-point-iterative-like method is applied to solve the linear system. [Figure 8: see original paper] illustrates the physical and mathematical meaning. Mathematically, $\mathbf{Ax} = \mathbf{Q}$ is transformed to $\mathbf{Mx} = \mathbf{Q} + \mathbf{Nx}$, where $\mathbf{Nx}$ represents the inflow terms of each element physically. Through point iteration, the exact linear system solution is obtained. Physically, all grid inflow terms are placed on the RHS, using lagged information among grids. Each grid is assigned to a GPU thread

for computation, making the new linear system particularly suitable for GPU calculation and yielding significant acceleration. Additionally, as mentioned in Section 3.2, the inflow term calculation kernel is unsuitable for GPUs. In the new linear system, this function is isolated as **Nx** and can be executed on CPU architecture simultaneously with GPU calculation. The CPU-GPU concurrent computing algorithm flowchart is shown in [Figure 9: see original paper]. In this algorithm, scattering source terms are computed on the GPU while inflow terms are computed on the CPU simultaneously. Based on GPU asynchronous streaming operations and MPS, angular parallelism is realized. For each angle, inflow terms are sent from CPU to GPU, the sparse linear system is solved, and related information is sent from GPU to CPU. Data transmission and computation are performed simultaneously, improving GPU performance. Compared with GPU acceleration algorithms in Sections 3.2 and 3.3, the new algorithm utilizes both CPU and GPU resources. Massive GPU threads are fully utilized in the new large-scale linear system form, and the sparse linear system is solved through simple matrix operations without limiting total GPU utilization.

[Figure 8: see original paper] Physical and mathematical meaning of the new method
[Figure 9: see original paper] Flowchart of CPU-GPU concurrent computing algorithm

4. Performance and Discussions

Typical PWR neutronics problems including the 2D TWIGL [43] and 2D C5G7 benchmarks [44] are used to evaluate algorithm performance. Six algorithms are tested: (1) Gauss-Seidel CPU Algorithm 1, (2) block Jacobi CPU Algorithm 2, (3) Gauss-Seidel GPU acceleration Algorithm 3 ([Figure 3: see original paper]), (4) block Jacobi GPU acceleration Algorithm 4 ([Figure 4: see original paper]), (5) linear system form GPU acceleration Algorithm 5 ([Figure 7: see original paper]), and (6) CPU-GPU concurrent computing Algorithm 6 ([Figure 9: see original paper]). Unless otherwise specified, first-order DFEM with triangular mesh discretization is applied, S4 Gauss-Legendre quadrature set is used, and convergence criteria are set to eigenvalue error $< 10^{-6}$ and scalar flux error $< 10^{-5}$.

The experimental platform consists of a 3.7 GHz Intel CPU (i5-12600KF) and an NVIDIA RTX-4070 Super GPU (7168 CUDA cores, 2.48 GHz) compiled under CUDA version 12.6.

4.1 2D TWIGL Benchmark

The geometry of the 2D TWIGL seed-blanket reactor benchmark and its discretization and domain decomposition are shown in [Figure 10: see original paper]. The TWIGL benchmark is a 160 cm $\times$ 160 cm square geometry with three different fuel regions using a two-group energy structure. Four vacuum boundary conditions are applied. Based on maximum grid size limitations, the

problem is divided into 2,162 to 101,758 grids, with the entire domain distributed across 16 CPU processors.

[Figure 10: see original paper] The discretization and domain decomposition of the TWIGL benchmark

Algorithm performance under varying grid numbers is tested, with results shown in . GPU memory usage and computation time for all six algorithms are listed. Comparing CPU Algorithms 1 and 2, block Jacobi CPU Algorithm 2 reduces total computation time by approximately 30% compared to Algorithm 1 for the TWIGL benchmark. General GPU Algorithms 3 and 4 achieve 4–5× speedup over CPU algorithms. Algorithm 4 uses less computation time than Algorithm 3 but requires slightly more GPU memory. Algorithm 4 treats boundary angular flux as calculable during sweeping, requiring more GPU memory than Algorithm 3. Linear system form GPU Algorithm 5 achieves approximately 1.5× speedup over general GPU Algorithm 4. Although more threads can be utilized in Algorithm 5, the serial computational process in restarted GMRES limits total utilization, yielding only modest acceleration over Algorithm 4. shows that CPU-GPU concurrent computing Algorithm 6 achieves the shortest computation time, delivering 24–60× speedup over CPU Algorithm 2 and 6–11× speedup over general GPU Algorithm 4. Algorithm 6 stores angular flux, source terms, and inflow terms, requiring the most GPU memory among all algorithms. As grid numbers increase from 2,162 to 8,474, Algorithm 6' s speedup increases from 6× to 11×, but as grids continue increasing, speedup gradually decreases to approximately 8×.

The performance of algorithms under different discretized grid numbers

4.2 2D C5G7 Benchmark

To further verify the concurrent computing algorithm' s efficiency, the more complex 2D C5G7 benchmark is tested. The geometry and discretization are shown in [Figure 11: see original paper]. The C5G7 benchmark is a heterogeneous transport problem with control rods published by OECD/NEA [44]. The 62.46 cm × 62.46 cm model is partitioned into 126,244 triangular meshes using an S8 quadrature set. The reactor core consists of four assemblies (two UO_2 and two MOX) surrounded by moderator. Both UO_2 and MOX assemblies follow a 17$×$17 configuration with 264 fuel pins, 2 guide tubes, and 1 fission chamber. Pin pitch is 1.26 cm × 1.26 cm with a cell radius of 0.54 cm. North and west boundaries are reflective; south and east boundaries are vacuum. The benchmark uses a seven-group energy library with detailed parameters in reference [44]. Numerical results are shown in . The CPU-GPU concurrent computing algorithm again achieves the highest speedup: 33× faster than CPU Algorithm 2 and 6.5× faster than general GPU Algorithm 4. Compared with TWIGL results, the concurrent computing algorithm's speedup slightly decreases, possibly due to three factors: (1) Algorithm 6' s thread requirements may exceed physical GPU threads, exceeding SM processing capacity and causing performance

degradation; (2) iterative format degradation from using delayed boundary information increases iteration count; (3) CPU capability for calculating inflow terms may be insufficient for larger-scale problems, increasing overall computation time.

[Figure 11: see original paper] The geometry and mesh partition of the 2D C5G7 benchmark
The performance of algorithms in C5G7 benchmark

5. Conclusions

General GPU acceleration algorithms for particle transport with unstructured grids are limited by wavefront dependence and cannot fully utilize GPU capability. To address this fundamental problem, the sweeping form of particle transport calculation is transformed into an equivalent linear system form. However, Krylov subspace methods like restarted GMRES for solving linear systems contain sequential and logical branch operations that reduce total GPU utilization. To solve these problems and leverage modern many-core CPU and GPU architectures, this paper proposes a CPU-GPU concurrent computing algorithm. The new algorithm utilizes more GPU threads, enables simultaneous CPU and GPU computation, and performs simultaneous data transmission and GPU computation. It achieves significant acceleration in typical neutronics TWIGL and C5G7 benchmarks, delivering 24–60× speedup over CPU algorithms and 6–11× speedup over general GPU algorithms.

Future work includes: (1) investigating methods to improve convergence and further accelerate the concurrent computing algorithm; (2) conducting theoretical convergence analysis; (3) addressing the challenging problem of CPU-GPU task allocation for large-scale simulations on heterogeneous CPU/GPU supercomputers.

References

[1] Chunye Gong, Jie Liu, et al. Particle transport with unstructured grid on GPU. *Computer physics Communications*, 183, 588-593, 2012. https://doi.org/10.1016/j.cpc.2011.12.002.
[2] C.Danani, H.L. Swami, et al. Advancements in computational methods for neutron shielding. *Micro and Nanostructured Composite Materials for Neutron Shielding Applications*, 379-399, 2020. https://doi.org/10.1016/B978-0-12-819459-1.00014-3.
[3] John Tramm, Bryce Allen, et al. Efficient algorithms for Monte Carlo particle transport on AI acceleration hardware. *Computer Physics Communications*, 109072,2024. https://doi.org/10.1016/j.cpc.2023.109072 .
[4] Benjamin Collins, Shane Stimpson, et al. Stability and accuracy of 3D neutron transport simulations using the 2D/1D method in MPACT. *Journal of Computational Physics*, 326, 612-628, 2016. https://doi.org/10.1016/j.jcp.2016.08.022.
[5] Cong Liu, Bin Zhang, et al. Verification of multilevel octree grid algorithm

of SN transport calculation with the Balakovo-3 VVER-1000 neutron dosimetry benchmark. *Nuclear Science and Technology*, 55, 756-768, 2023. https://doi.org/10.1016/j.net.2022.10.017.
[6] Shuangshuang Chen. A combined mixed finite element method and discontinuous Galerkin method hybrid-dimensional fracture models of two-phase flow. *Journal of Computational and Applied Mathematics*, 495, 116373,2015. https://doi.org/10.1016/j.cam.2024.116373.
[7] G.Colomer, R. Borrell, et al. Parallel algorithm for SN transport sweeps on unstructured meshes. *Journal of Computational Physics*, 232, 118-135, 2013. https://doi.org/10.1016/j.jcp.2012.07.009.
[8] S. Plimpton, B. Hendrickson, et al. Parallel algorithms for radiation transport on unstructured grids. *Proceedings ACM Conference*, Dallas, Texas, https://doi.org/10.1109/SC.2000.10030.
[9] Zhiwei Zong, Maosong Cheng, et al. A multithreaded parallel upwind sweep algorithm for the SN transport equations discretized with discontinuous finite elements. *Nuclear Science and Techniques*, 34, 200, 2023. https://doi.org/10.1007/s41365-023-01355-4.
[10] S.D. Pautz. An algorithm for parallel SN sweeps on unstructured meshes. *Nuclear Science and Engineering*, 140, 11-136, 2002. https://doi.org/10.13182/NSE02-1.
[11] Mo Zeyao, Fu Lianxiang. Parallel flux sweep algorithm for neutron transport on unstructured grid. *The Journal of Supercomputing*, 30,5-17,2004. https://doi.org/10.1023/B:SUPE.000032778.36178.d8.
[12] Jan I.C. Vermaak, Jean C. Ragusa, et al. Massively parallel transport sweeps on meshes with cyclic dependencies. *Journal Computational Physics*, 109892, https://doi.org/10.1016/j.jcp.2020.109892.
[13] C. Liu, E.W. Larsen. Discrete ordinates methods for the transport problems with curved spatial grids. PhD dissertation, University of Michigan, Department of Nuclear Engineering and Radiological Sciences, 2015.
[14] T.S. Haut, P.G. Maginot, et al. An efficient sweep-based solver for the SN equations on high-order meshes. *Nuclear Science Engineering*, 193:7, https://doi.org/10.1080/00295639.2018.1562778.
[15] D.S. Efremenko, et al. Multi-core-CPU and GPU accelerated radiative transport models based on discrete ordinate method. *Computer Physics Communications*, 07,018,2014. https://doi.org/10.1016/j.cpc.2014.07.018.
[16] Weiguang Li, Cheng Chang, et al. GPU based cross-platform Monte Carlo proton dose calculation engine framework of Taichi. *Nuclear Science and Techniques*, 34,77, https://doi.org/10.1007/s41365-023-012118-y.
[17] Andrey Gorobets, Pavel Bakhvalov. Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressive turbulent flows on hybrid supercomputers. *Computer Physics and Communications*, 271, 108231, 2022. https://doi.org/10.1016/j.cpc.2021.108231.
[18] Chunye Gong, Jie Liu, et al. GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method. *Journal of Computational Physics*, 230, 6010-6022, 2011. https://doi.org/10.1016/j.jcp.2011.04.0110.
[19] Yaqi Wang, Jean C. Ragusa. A high-order discontinuous Galerkin method

for the SN transport equations on 2D unstructured triangular meshes. *Annals of Nuclear Energy*, 36(7), 931-939, 2007. https://doi.org/10.1016/j.anucene.2009.03.002.
[20] Fengshun Lu, Junqiang Song, et al. Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters. *Computer Physics Communications*, 138 (6), 1172-1181, 2012. https://doi.org/10.1016/j.cpc.2012.01.019.
[21] Feiyu Chen, Ganglin Yu, et al. Parallelization and optimization of RMC for criticality computing based on heterogeneous architecture of the Sunway TaihuLight supercomputer. *Annals of Nuclear Energy*, 147, 107761, 2020. https://doi.org/10.1016/j.anucene.2020.107761.
[22] NVIDIA Corporation, *CUDA Programming Guide*, version 12.6, 2024.
[23] Boran Kong, Kaijie Zhu, et al. Assessment of computational technicalities for the 2D/1D coupling method and its comparison with 3D MOC. *Science and Technology of Nuclear Installations*, 19, 2198653, 2024. https://doi.org/10.1155/2024/2198653.
[24] Liang Qiao, Youqi Zheng, et al. Improved block-Jacobi parallel algorithm for the SN nodal method with unstructured mesh. *Progress in Nuclear Energy*, 103629, https://doi.org/10.1016/j.pnucene.2021.103629.
[25] Boran Kong, Kaijie Zhu, et al. A discontinuous Galerkin finite element method based axial SN for 2D/1D transport method. *Progress in Nuclear Energy*, 104391, https://doi.org/10.1016/j.pnucene.2022.104391.
[26] D.P. Truong, M.I. Ortega et al. Tensor networks for solving the time-independent Boltzmann neutron transport equation. *Journal Computational Physics*, 507(15),112943. https://doi.org/10.1016/j.jcp.2024.112943.
[27] B.D. Rodriguez, M.T. Vilhena, B.E.J Bodmann. An overview of the Boltzmann transport equation solution for neutrons, photons and electrons in Cartesian geometry. *Progress in Nuclear Energy*, 53 (8), 1119-1125, 2011. https://doi.org/10.1016/j.pnucene.2011.06.009.
[28] Boran Kong, Kaijie Zhu, et al. Incorporation of anisotropic scattering into Fourier moment expanded axial DGFEM SN based 2D/1D coupling method. *Annals of Nuclear Energy*, 192, 109955, 2023. https://doi.org/10.1016/j.anucene.2023.109955.
[29] Erin D. Fichtl, James S. Warsa et al. The Newton-Krylov method applied to negative flux fixup in SN transport calculations. *Nuclear Science and Engineering*, 165:3, 331-341, https://doi.org/10.13182/NSE09-51.
[30] Peter G. Maginot, Jean C. Ragusa, et al. Discontinuous finite element discretization for the SN neutron transport equation in problems with spatially varying cross sections. *Annals of Nuclear Energy*, 73, 506-526, 2014. https://doi.org/10.1016/j.anucene.2014.06.050.
[31] Haoju Lin, et al. A parallel parameterized level set topology optimization framework for large-scale structures with unstructured meshes. *Computer Method in Applied Mechanics and Engineering*, 397,115112, 2022. https://doi.org/10.1016/j.cma.2022.115112.
[32] S.D.Pautz, et al. Parallel deterministic transport sweeps of structured and unstructured meshes with overloaded mesh decomposition. *Nuclear Science and Engineering*, 185, 70-77, 2017. https://doi.org/10.13182/NSE16-34.
[33] Bowen Xiao, Zhe Wang, et al. The probability calculation of a persistent fis-

sion chain based on discontinuous Galerkin finite element method. *Nuclear Engineering and Design*, 433, 113848, 2025. https://doi.org/10.1016/j.nucengdes.2025.113848.
[34] L.S. Dominque, M.M. Patwary. Improved graph partitioning for modern graphs and architectures. 5th workshop on irregular applications: architectures and algorithms, supercomputing, 2015.
[35] Daniel Nagy, Laszlo Pasthy, et al. Development of a GPU-based DEM solver for parameter optimization in the simulations of soil-sweep tool interaction. *Computers and Electronics in Agriculture*, 227(1), 109482, 2024. https://doi.org/10.1016/j.compag.2024.109482.
[36] Xinxin Mei, Kaiyong Zhao, et al. Benchmarking the memory hierarchy of modern GPUs. *Network and Parallel Computing*, NPC2014. https://doi.org/10.1007/978-3-662-44917-2_{13}.
[37] Xinxin Mei, Xiaowen Chu, et al. Dissecting GPU memory hierarchy through microbenchmarking. *Transactions Parallel Distributed Systems*, https://doi.org/10.1109/TPDS.2016.2549523.
[38] Bing Hu. Research on application of GPU parallel computing and CUDA programming in environmental engineering. Master dissertation, China University of Geosciences, Beijing, 2017.
[39] Dai Ni, Dai Tao, et al. Negative flux fixups using positivity-preserving limiters for SN discontinuous finite element scheme of neutron transport equation on unstructured triangular meshes. *Annals of Nuclear Energy*, 211, 111025,2025. https://doi.org/10.1016/j.anucene.2024.111025.
[40] Pedro Cortez Lopes, Federico Semeraro, et al. Enabling FEM-based absolute permeability estimation in giga-voxel porous media with a single GPU. *Computer Methods in Applied Mechanics and Engineering*, 434, 117559, 2025. https://doi.org/10.1016/j.cma.2024.117559.
[41] Bo Yang, Hui Liu, et al. Preconditioned GMRES solver on multiple-GPU architecture. *Computers Mathematics Applications*, 1076-1095, https://doi.org/10.1016/j.cmawa.2016.06.027.
[42] Qinmeng Zou. GMRES algorithm over 35 years. *Applied Mathematics and Computation*, 445, 127869, 2023. https://doi.org/10.1016/j.cma.2023.127869.
[43] L.A. Hageman, J.B. Yasinsky. Comparison of alternating-direction time-differencing methods with other implicit methods for the solution of the neutron group-diffusion equations. *Nuclear Science and Engineering*, 28, 2017. https://doi.org/10.13182/NSE38-8.
[44] Smith M.A., et al. Benchmark on deterministic transport calculations without homogenization. *Nuclear Energy Agency Organization for Economic Co-operation and Development*, 2003.

Note: Figure translations are in progress. See original paper for figures.