CPU-GPU concurrent computing algorithm of particle transport using discontinuous finite element discrete ordinates with unstructured grids

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

Date: 2025-04-14T16:01:11+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) approach for solving particle transport problems represents a substantial computational burden, consuming significant calculation time in multi-physics simulations. The emergence of graphics processing units (GPUs) has transformed computing capabilities, offering new avenues for DFEM-SN calculations. However, conventional GPU acceleration algorithms for particle transport on unstructured grids suffer from limited effectiveness due to dependencies in the SN spatial wavefront, preventing full utilization of GPU computing power. This paper proposes a CPU-GPU concurrent computing algorithm that formulates the problem as a large-scale linear system solved with a block Jacobi parallel strategy. In this algorithm, fission and scattering source terms are computed on the GPU while inflow terms undergo coarse-grained domain calculations on the CPU simultaneously. Furthermore, decomposition parallelism and fine-grained angular parallelism are employed, enabling simultaneous data transmission and computation. This novel algorithm not only leverages the massive thread count available 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 acceleration 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 [**?**]. 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 [**?**, **?**]. The discrete ordinates (SN) method is widely employed for solving the transport equation by selecting representative discrete angles for angular discretization [**?**]. Meanwhile, to accommodate increasingly complex geometric designs and to handle large deformation problems effectively, SN methods frequently adopt the discontinuous finite element method (DFEM) for spatial discretization [**?**]. However, DFEM-SN approaches for particle transport impose substantial demands on both memory requirements and computational time [**?**].

With the advancement of high-performance computing (HPC), extensive re-

search has been conducted on DFEM-SN parallelization for unstructured grids. Plimpton proposed an asynchronous algorithm utilizing grid partitioning and cycle detection [**?**], while Mo developed a parallel flux sweep algorithm based on domain decomposition [**?**]. The fundamental principle underlying these algorithms involves converting the element sweep order into a directed acyclic graph (DAG) [**?**]. Numerous efforts have focused on handling cyclic dependencies during transport sweeps [**?**] and improving parallel efficiency on central processing units (CPUs) [**?**]. On the other hand, heterogeneous CPU-GPU systems have advanced rapidly [**?**], providing new methodologies for high-fidelity DFEM-SN transport calculations. Zhang et al. proposed a GPU-based multi-group discrete ordinates transport calculation and developed the STRAUM code [**?**], while 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 [**?**, **?**]. In these simulations, however, data dependencies across wavefronts [**?**] limit parallelism and prevent full utilization of GPU computing capabilities.

Modern HPC architectures incorporate many-core CPUs and GPUs, enabling simultaneous execution on both processor types [**?**, **?**]. To circumvent the inherent parallelism limitations of wavefront sweeping and to adapt to contemporary computing architectures, we propose a CPU-GPU concurrent computing DFEM-SN algorithm. The algorithm is formulated as a large-scale linear system and employs a hybrid Message Passing Interface (MPI) [**?**] programming model. Domain and Compute Unified Device Architecture (CUDA) decomposition [**?**] are implemented as coarse-grained parallelism, with each subdomain assigned to a specific CPU processor. The block Jacobi parallel strategy [**?**] is then applied: source terms are calculated on GPUs while inflow terms are computed simultaneously on CPUs using coarse-grained domain decomposition. Angular parallelism is implemented as fine-grained parallelism based on GPU asynchronous streaming operations and Multi-Process Service (MPS) [**?**]. Compared with traditional DAG-based CPU algorithms and general GPU acceleration approaches, this novel CPU-GPU concurrent algorithm achieves significant speedup in typical pressurized water reactor (PWR) [**?**] 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 evaluates the performance of various algorithms on typical PWR benchmarks and provides related discussions. Section 5 summarizes the conclusions.

2.1 SN Transport Equation

Particle transport through various media is described by the Boltzmann equation, derived from particle balance conditions [**?**, **?**]. Using neutrons as an example (the steady-state Boltzmann equation for other particles such as photons

and electrons follows a similar form):

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

where $\psi(\mathbf{r},E,\ )$ represents the angular flux as a function of position $\mathbf{r}$, energy E, and direction . The scalar flux 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 neutrons per fission. For numerical solution via deterministic methods, the basic problem is discretized in space, angle, and energy [**?**, **?**].

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

$$_m\cdot\nabla\psi_{g,m}(\mathbf{r})+\Sigma_{t,g}(\mathbf{r})\psi_{g,m}(\mathbf{r})=\frac{\Sigma_{s,g}(\mathbf{r})}{4\pi}\phi_g(\mathbf{r})+\frac{\chi_g(\mathbf{r})}{k_{\text{eff}}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(\mathbf{r})\phi_{g'}(\mathbf{r})$$

where subscript m denotes the angular index and g denotes 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 provides stability and accuracy for solving first-order equations [**?**]. DFEM integrates Eq. (2) over elements with weighting functions $w(\mathbf{r})$:

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

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

$$\int_{\partial V} w(\mathbf{r})\ _m\cdot\mathbf{n}\psi_{g,m}(\mathbf{r})\,dS-\int_V\nabla w(\mathbf{r})\cdot\ _m\psi_{g,m}(\mathbf{r})\,dV+\int_V w(\mathbf{r})\Sigma_{t,g}(\mathbf{r})\psi_{g,m}(\mathbf{r})\,dV=\int_V w(\mathbf{r})Q_{g,m}(\mathbf{r})\,dV$$

Unlike continuous FEM, DFEM permits discontinuities in the surface integration term, as shown in the first term of Eq. (5):

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

The superscript + indicates quantities outside the element surface, while − indicates quantities inside. DFEM here specifically refers to the linear discontinuous Galerkin finite element method, where weight functions are identical to trial functions [**?**]. The angular flux and source term are linearly expanded using Nth-order basis functions:

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

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

$$\mathbf{A}^e_{g,m}\psi^e_{g,m}=\mathbf{Q}^e_{g,m}+\mathbf{B}^{e,\text{neighbor}}_{g,m}\psi^{\text{neighbor}}_{g,m}$$

where

$$\mathbf{A}^e_{g,m}=\int_V\left[\Sigma_{t,g}(\mathbf{r})\mathbf{b}(\mathbf{r})\mathbf{b}^T(\mathbf{r})-\ {}_m\cdot\nabla\mathbf{b}(\mathbf{r})\mathbf{b}^T(\mathbf{r})\right]dV+\int_{\partial V^+}\ {}_m\cdot\mathbf{n}\mathbf{b}(\mathbf{r})\mathbf{b}^T(\mathbf{r})\,dS$$

$$\mathbf{B}^{e,\text{neighbor}}_{g,m}=\int_{\partial V^-}\ {}_m\cdot\mathbf{n}\mathbf{b}(\mathbf{r})\mathbf{b}^T(\mathbf{r})\,dS$$

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

2.3 Parallel Strategies

The DFEM-SN equation in Eq. (8) is typically solved iteratively, involving 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 [**?**, **?**]. The energy loop typically proceeds from high to low energy groups to facilitate updating of down-scattering terms [**?**, **?**]. The angular loop has no specific sequence requirements under 2D Cartesian coordinates. For a given

direction, the sweeping sequence on unstructured grids can be determined using a DAG, as illustrated in Fig. 1. Each node represents a triangular grid element, and arrows between nodes denote dependency relationships. The DAG defines cells by mapping each grid element to each angular direction, with cell in-degree defined as the sum of inflow contributions to that cell [**?**]. Detailed DAG theory can be found in Refs. [**?**].

Algorithm 1: DFEM-SN Iteration Calculation

```
While (Not converged) Outer iteration {
    For (g=1:G) energy loop {
        While (Not converged || Reach specific inner iteration limit) Inner iteration {
            Update scattering source term
            For (m=1:M) angle loop {
                Sweep over entire spatial 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

As shown in Fig. 1, the entire computational domain is partitioned into several subdomains and 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 commonly employed: Gauss-Seidel and block Jacobi. In the Gauss-Seidel strategy, after receiving incoming messages from other processors, the in-degree of relevant boundary cells is decremented. When the in-degree reaches zero, the boundary cell becomes calculable and subsequent computations can proceed. In the block Jacobi strategy, boundary cells utilize delayed boundary information from other processors, with the in-degree from external 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 necessitates continuous message passing during sweeping. Block Jacobi' s use of delayed boundary information reduces communication and waiting time at the cost of increased outer iterations [**?**].

To improve block Jacobi convergence, energy group looping is commonly applied in practice. Cell 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, thereby abandoning the energy group loop. Con-

sequently, the iterative structure differs between the two strategies: block Jacobi follows Algorithm 1, while Gauss-Seidel follows Algorithm 2. After an inner iteration, Gauss-Seidel updates scattering terms for all energy groups simultaneously. 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 across inner iterations, allowing pre-calculation and storage of sweep orders to reduce DAG determination overhead. In contrast, Gauss-Seidel must determine the sweeping sequence online for each inner iteration, representing a non-trivial computational burden. Both block Jacobi and Gauss-Seidel strategies have distinct advantages and are employed in neutron transport calculations, with numerous optimization measures achieving significant improvements [**?**, **?**], though these are beyond the scope of this paper.

Algorithm 2: Gauss-Seidel Iteration Calculation

```
While (Not converged) Outer iteration {
    While (Not converged || Reach specific inner iteration limit) Inner iteration {
        Update scattering source term for 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, we developed the DFEM-SN code SNIPER [**?**]. SNIPER is written in C++ and utilizes a hybrid MPI and CUDA programming model. The framework of the steady-state SNIPER code is shown in Fig. 2. The code employs the mature graph partitioning software METIS from Karypis Lab for domain decomposition [**?**]. Control parameters, material properties, and iteration indices are specified and modified through input and material cards. Flux distributions, geometry, and material information are verified and displayed via output cards. The code is organized into several classes: transport information, mesh information, sweeping, integration, parallelization, and calculation. The transport information class configures basic parameters such as cross-sections and quadrature sets based on input specifications. The mesh information class reads unstructured meshes from msh-format mesh files and interfaces with METIS for domain decomposition. The parallel class accumulates parallel performance metrics 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, whereas 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 [**?**]. 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 [**?**, **?**]. 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 handles task-level macro-operations among SMs, while fine-grained parallelism performs specific data operations within a single SM, delivering high parallel performance [**?**]. In summary, CPUs with their sophisticated control logic and large caches are suited for general-purpose computing and complex logical operations, while GPUs leverage massive thread counts for large-scale parallel tasks with high computational density and simple control flow.

For realistic problems, solving the DFEM-SN equation (8) represents a massive computational burden requiring over 10^{13} degrees of freedom [**?**, **?**]. GPUs provide a new approach for solving Eq. (8), primarily by accelerating large-scale computations. Flowcharts of general GPU acceleration for unstructured grid neutron transport are shown in Fig. 3 and Fig. 4. Fig. 3 illustrates the Gauss-Seidel parallel strategy in the MPI-CUDA programming model, which employs pipeline SN sweeping to prevent iteration degradation, requiring frequent CPU-GPU communication. The GPU acceleration algorithm for the block Jacobi parallel strategy is shown in Fig. 4, where updated information is communicated uniformly 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 utilized 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 well-suited for parallel computation and straightforward to implement on GPUs. The inflow term calculation in Eq. (9) uses neighbor angular flux values and requires branch operations including determination of CPU ownership and reflective boundary conditions. For DFEM with iso-parametric elements [**?**], neighbor side ownership must also be determined. Such extensive branching reduces GPU efficiency. Fig. 5 shows the number of calculable elements per sweeping step using the block Jacobi parallel strategy for a typical PWR neutronics benchmark. The GPU thread count for kernels 2 and 3 is determined by the number of calculable elements,

which varies dramatically from a maximum of 914 to a minimum of only 4. The number of thread blocks changes substantially across sweeping steps, indicating that GPU computing capability is not fully utilized. Wavefront dependencies in the sweeping sequence limit parallelism and represent the bottleneck for GPU acceleration [**?**, **?**].

[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 [**?**, **?**]. Therefore, we designed a new GPU acceleration algorithm formulated as a large-scale linear system.

The DFEM-SN equation (8) can be expressed in both linear system and sweeping forms, as illustrated in Fig. 6. Equation (8) for all elements in the entire domain is assembled into a large-scale linear system $\mathbf{Ax} = \mathbf{Q}$, where $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, ...]^T$ represents the solution vector, with $\mathbf{x}_1$ containing all unknowns of the first grid and so forth. It should be noted that the points in $\mathbf{x}$ are not single variables but sets of all unknowns for each grid. Based on DAG analysis, the first and second grids can be computed first. Due to inter-grid dependencies, the third and fourth grids are calculated sequentially. The sweeping form can be understood as a physical method for solving this large-scale linear system, representing a theoretically optimized approach with clear physical meaning. The linear system is sparse and can be solved using Krylov subspace methods such as the Generalized Minimal Residual (GMRES) algorithm [**?**, **?**]. Although GMRES loses the physical meaning of sweeping, it conceals element dependencies in the sweeping sequence, enabling full utilization of massive GPU threads and offering a viable approach for solving Eq. (8) on GPU architectures.

Based on this analysis, we propose a new GPU acceleration algorithm shown in Fig. 7. The entire computational subdomain for each discrete angle is uniformly written into sparse linear system form. Boundary inflow terms from other CPUs are treated as known variables and added to the RHS of the linear system, which is 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 the preparation phase. Traditional GMRES algorithms increase subspace dimension and data storage during computation, which are difficult operations for GPUs. Therefore, we employ the restarted GMRES algorithm with fixed subspace dimension [**?**] to solve the sparse linear system. Unlike general GPU

acceleration algorithms, the inflow terms and element dependencies are embedded within the sparse linear system, and restarted GMRES calculations for all discrete 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 in this algorithm: (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) involves checking for GMRES restart termination, a branch operation inefficient on GPUs; and (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 previous 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 $i = 1 : j$ { $h_{i,j} = \mathbf{w}^T\mathbf{v}_i$ $\mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$ } $h_{j+1,j} = \|\mathbf{w}\|_2$ If $h_{j+1,j} = 0$, break to step 4 $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$ }
4. Solve least squares problem $\min \|\beta\mathbf{e}_1 - \bar{H}_j\mathbf{y}\|_2$ via Givens rotation, $\mathbf{x}_j = \mathbf{x}_0 + V_j\mathbf{y}$
5. If $\|\mathbf{r}_j\|_2 <$ tol, solution found; else set $\mathbf{x}_0 = \mathbf{x}_j$ 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 the restarted GMRES solver limits its total GPU utilization. Meanwhile, modern HPC architectures incorporate many-core CPUs and GPUs, enabling simultaneous execution on both processors. Inspired by these considerations, we propose a CPU-GPU concurrent computing algorithm.

Similar to Section 3.3, we maintain the large-scale linear system formulation of the transport calculation. Non-diagonal block elements in matrix $\mathbf{A}$ are rearranged onto the RHS of the linear system. Instead of Krylov subspace methods, we apply a block-Jacobi-point-iterative-like approach to solve the linear system. Fig. 8 illustrates the physical and mathematical interpretation: mathematically, $\mathbf{Ax} = \mathbf{Q}$ is transformed to $\mathbf{Mx} = \mathbf{Q} + \mathbf{Nx}$, where $\mathbf{Nx}$ represents the inflow terms of each element in physical terms. Through point iteration, the exact solution of the linear system is obtained. Physically, inflow terms for all grids are placed on the RHS and use lagged information from other grids. Each

grid is assigned to a GPU thread for computation, making the new linear system format particularly suitable for GPU calculation and yielding significant acceleration. Additionally, as noted in Section 3.2, the inflow term calculation kernel is not well-suited for GPUs. In the new linear system, this function is isolated as **Nx** and can be executed on CPU architecture simultaneously with GPU computation. The CPU-GPU concurrent computing algorithm flowchart is shown in Fig. 9. In this algorithm, scattering source terms are computed on the GPU while inflow terms are computed on the CPU simultaneously. Angular parallelism is achieved through GPU asynchronous streaming operations and Multi-Process Service (MPS). For each angle, inflow terms are sent from CPU to GPU, the sparse linear system is solved, and relevant information is returned from GPU to CPU. Data transmission and computation are performed concurrently, thereby improving GPU performance. Compared with the GPU acceleration algorithms in Sections 3.2 and 3.3, the new algorithm utilizes both CPU and GPU resources. Massive GPU threads are fully exploited through the new large-scale linear system format, where the sparse linear system is solved via 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 [**?**] and 2D C5G7 [**?**] benchmarks are used to evaluate algorithm performance. Six algorithms are tested: (1) Gauss-Seidel CPU Algorithm, (2) block Jacobi CPU Algorithm, (3) Gauss-Seidel GPU acceleration Algorithm (Fig. 3), (4) block Jacobi GPU acceleration Algorithm (Fig. 4), (5) linear system form GPU acceleration Algorithm (Fig. 7), and (6) CPU-GPU concurrent computing Algorithm (Fig. 9). Unless otherwise specified, first-order DFEM with triangular mesh discretization is applied, S4 Gauss-Legendre quadrature 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 Fig. 10. The TWIGL benchmark features a 160 cm $\times$ 160 cm square geometry with three distinct fuel regions and a two-group energy structure. Four vacuum boundary conditions are applied. Based on grid size limitations, the problem is discretized 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 Table 1. GPU memory usage and computation time for all six algorithms are listed. Comparing CPU Algorithms 1 and 2 reveals that 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, as boundary angular fluxes are treated as calculable during sweeping, increasing memory requirements. 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, resulting in only modest acceleration over Algorithm 4. Table 1 demonstrates 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 acceleration 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 acceleration effect improves from 6× to 11×, but further grid increases gradually reduce the acceleration to approximately 8×.

The performance of algorithms under different discretized grid numbers

4.2 2D C5G7 Benchmark

To further verify the efficiency of the CPU-GPU concurrent computing algorithm, the more complex 2D C5G7 benchmark is tested. The geometry and discretization of the C5G7 benchmark are shown in Fig. 11. The C5G7 benchmark is a heterogeneous transport problem with control rods published by OECD/NEA [**?**]. The model size is 62.46 cm × 62.46 cm, partitioned into 126,244 triangular meshes with an S8 quadrature set applied. 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 available in reference [**?**]. Numerical results are shown in Table 2, demonstrating that the CPU-GPU concurrent computing algorithm again achieves the highest acceleration: 33× faster than CPU Algorithm 2 and 6.5× faster than general GPU acceleration Algorithm 4. Compared with TWIGL results, the concurrent computing algorithm' s acceleration effect is slightly reduced, likely due to three factors: (1) Algorithm 6' s thread requirements may exceed physical GPU threads, overloading SMs and

causing performance degradation; (2) iterative format degradation from using delayed boundary information increases iteration count; and (3) CPU capability for inflow term calculation may become 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 on unstructured grids are limited by wavefront dependencies and cannot fully utilize GPU capabilities. To address this fundamental issue, the sweeping form of particle transport calculation is transformed into an equivalent linear system form. However, Krylov subspace methods like restarted GMRES for solving the linear system contain sequential and logical branch operations that reduce total GPU utilization. To overcome these limitations and leverage modern many-core CPU-GPU architectures, this paper proposes a CPU-GPU concurrent computing algorithm. The new algorithm utilizes more GPU threads, enables simultaneous CPU-GPU computation, and performs data transmission and computation concurrently. It achieves significant acceleration in typical neutronics benchmarks (TWIGL and C5G7), delivering 24-60× speedup over CPU algorithms and 6-11× speedup over general GPU algorithms.

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

References

[1] C.Y. Gong, J.Liu, H.W. Huang et al. Particle transport with unstructured grid on GPU. Comp. Phys. Comm. 183, 588-593 (2012). https://doi.org/10.1016/j.cpc.2011.12.002.
[2] H. Yin, X.J. Liu, T.F. Zhang et al. An efficient parallel algorithm of variational nodal method for heterogeneous neutron transport problems. Nucl. Sci. Tech.35, 69 (2024). https://doi.org/10.1007/s41365-024-01430-4.
[3] E.P. Zhang, Z.N. Zhang, N.N. Zhang et al. Development and verification of the higher-order mode neutron flux calculation code HARMONY2.0. Nucl. Sci. Tech.36, 18 (2015). https://doi.org/10.1007/s41365-024-01619-7.
[4] B. Collins, S. Stimpson, B.W. Kelly et al. Stability and accuracy of 3D neutron transport simulations using the 2D/1D method in MPACT. Jour. Comp. Phys. 326, 612-628 (2016). https://doi.org/10.1016/j.jcp.2016.08.022.
[5] C. Liu, B. Zhang, J.X. Wei et al. Verification of multilevel octree grid algorithm of SN transport calculation with the Balakovo-3 VVER-1000

neutron dosimetry benchmark. Nucl. Eng. Tech. 55, 756-768 (2023). https://doi.org/10.1016/j.net.2022.10.017.
[6] N. Dai, B. Zhang, Y.X. Chen et al. Adaptive discontinuous finite element quadrature sets over an icosahedron for discrete ordinates method. Nucl. Sci. Tech. 32, 98 (2021). https://doi.org/10.1007/s41365-021-00934-7.
[7] G.Colomer, R. Borrell, F.X. Trias et al. Parallel algorithm for SN transport sweeps on unstructured meshes. Jour. of Comp. Phys. , 232, 118-135 (2013). https://doi.org/10.1016/j.jcp.2012.07.009.
[8] S. Plimpton, B. Hendrickson, S. Burns et al. Parallel algorithms for radiation transport on unstructured grids. Proceedings of the 2000 ACM Conference (2000). https://doi.org/10.1109/SC.2000.10030.
[9] Z.Y. Mo, L.X. Fu. Parallel flux sweep algorithm for neutron transport on unstructured grid. Jour. of Supercomputing. 30,5-17 (2004) . https://doi.org/10.1023/B:SUPE.000032778.36178.d8.
[10] S.D. Pautz. An algorithm for parallel SN sweeps on unstructured meshes. Nucl. Sci. Eng. 140, 11-136 (2002) . https://doi.org/10.13182/NSE02-1.
[11] Z.W. Zong, M.S. Cheng, Y.C. Yu et al. A multithreaded parallel upwind sweep algorithm for the SN transport equations discretized with discontinuous finite elements. Nucl. Sci. Tec. 34, 200 (2023). https://doi.org/10.1007/s41365-023-01355-4.
[12] T.S. Haut, P.G. Maginot, V.Z. Tomov et al. An efficient sweep-based solver for the SN equations on high-order meshes. Nucl. Sci. Eng. 7, 193 (2019). https://doi.org/10.1080/00295639.2018.1562778.
[13] D.S. Efremenko, D.G. Loyola, A. Doicu et al. Multi-core-CPU and GPU accelerated radiative transport models based on the discrete ordinate method. Comp. Phys. Comm. 07,018 (2014). https://doi.org/10.1016/j.cpc.2014.07.018.
[14] W.G. Li, C. Chang, Y. Qin et al. GPU based cross-platform Monte Carlo proton dose calculation engine in the framework of Taichi. Nucl. Sci. Tech. 34,77 (2023). https://doi.org/10.1007/s41365-023-012118-y.
[15] Y.H. Wang, L.M. Yan, B.Y. Xia et al. Lattice Boltzmann method for simulation of time-dependent neutral particle transport. Nucl. Sci. Tech. 28, 36 (2017). https://doi.org/10.1007/s41365-017-0185-z
[16] A. Gorobets, P. Bakhvalov. Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressive turbulent flows on hybrid supercomputers. Comp. Phys. Comm. 271, 108231 (2022). https://doi.org/10.1016/j.cpc.2021.108231.
[17] A. Zhang, S.G. Hong, S.J et al. GPU-based multi-group discrete ordinates transport calculations: Parallel computing implementation in STRAUM. Nucl. Eng. Tech. 103579 (2025). https://doi.org/10.1016/j.net.2025.103597
[18] C.Y. Gong, J. Liu, L.H. Chi et al. GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method. Jour. Comp. Phys. 230, 6010-6022 (2011). https://doi.org/10.1016/j.jcp.2011.04.0110.
[19] Y.Q. Wang, J.C. Ragusa. A high-order discontinuous Galerkin method for the SN transport equations on 2D unstructured triangular meshes. Ann. Nucl. Energy. 36, 31-939 (2007). https://doi.org/10.1016/j.anucene.2009.03.002.
[20] A.K. Hu, R. Qiu, H. Liu et al. THUBrachy: fast Monte Carlo dose

calculation tool accelerated by heterogeneous hardware for high-dose-rate brachytherapy. Nucl. Sci. Tech. 32, 32 (2021). https://doi.org/10.1007/s41365-021-00866-2
[21] X.Y. Luo, L. Sun, Z. Wu et al. gMCAP: a GPU-based Monte Carlo proton transport program for high-density tissues with precise nuclear reaction models. Nucl. Sci. Tech. 36, 83 (2025). https://doi.org/10.1007/s41365-025-01655-x
[22] NVIDIA Corporation, CUDA Programming Guide, version 12.6, 2024.
[23] B.R. Kong, K.J. Zhu, H. Zhang et al. Assessment of computational technicalities for the 2D/1D coupling method and its comparison with 3D MOC. Sci. Tech. Nucl. Inst.19, 2198653 (2024). https://doi.org/10.1155/2024/2198653.
[24] L. Qiao, Y.Q. Zheng, H.C. Wu et al. Improved block-Jacobi parallel algorithm for the SN nodal method with unstructured mesh. Prog. Nucl. Energy. 133, 103629 (2021). https://doi.org/10.1016/j.pnucene.2021.103629.
[25] B.R. Kong, K.J. Zhu, H. Zhang et al. A discontinuous Galerkin finite element method based axial SN for the 2D/1D transport method. Prog. Nucl. Energy. 152, 104391(2022). https://doi.org/10.1016/j.pnucene.2022.104391.
[26] K.M. Yassin, M.H. Hassan, M.M. Ghoneim et al. Multiphysics simulation of VVER-1200 fuel performance during normal operating conditions. Nucl. Sci. Tech. 34, 28 (2023). https://doi.org/10.1007/s41365-023-01172-9
[27] L.X. Liu, C. Hao, Y.L. Xu. Equivalent low-order angular flux nonlinear finite difference equation of MOC transport calculation. Nucl. Sci. Tech. 31, 125 (2020). https://doi.org/10.1007/s41365-020-00834-2
[28] B.R. Kong, K.J. Zhu, H. Zhang et al.Incorporation of anisotropic scattering into Fourier moment expanded axial DGFEM SN based 2D/1D coupling method. Annals Nucl. Energy. 192, 109955 (2023). https://doi.org/10.1016/j.anucene.2023.109955.
[29] L.Y. He, Y. Cui, L. Chen et al. Effect of reprocessing on neutrons of a molten chloride salt fast reactor. Nucl. Sci. Tech. 34, 46(2023). https://doi.org/10.1007/s41365-023-01186-3
[30] H. Zhang, D. Li, T.L. Hu et al. SphiNx: A SN-DFEM Neutron Transport Code Based on the MOOSE Framework. Nucl. Eng. Tech. 103641 (2025). https://doi.org/10.1016/j.net.2025.103641
[31]H.J. Lin, H. Liu, P. Wei. A parallel parameterized level set topology optimization framework for large-scale structures with unstructured meshes. Comp. Meth. Appl. Mech. Eng., 397,115112 (2022). https://doi.org/10.1016/j.cma.2022.115112.
[32] S.D.Pautz, T.S. Bailey. Parallel deterministic transport sweeps of structured and unstructured meshes with overloaded mesh decomposition. Nucl. Sci. Eng. 185, 70-77 (2017). https://doi.org/10.13182/NSE16-34.
[33] B.W. Xiao, Z. Wang, L.F. Xu et al. The probability calculation of a persistent fission chain based on discontinuous Galerkin finite element method. Nucl. Eng. Des. 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). https://dl.acm.org/doi/pdf/10.1145/2833179.2833188
[35] L. Wang, J.C. Yang, M.X. Chang et al. GOAT: a simulation code for high-

intensity beams. Nucl. Sci. Tech. 34, 78 (2023). https://doi.org/10.1007/s41365-023-01225-z
[36] P.T. Song, Z.Z. Zhang, Q. Zhang et al. Implementation of the CPU/GPU hybrid parallel method of characteristics neutron transport calculation using the heterogeneous cluster with dynamic workload assignment. Anna. Nucl. Energy. 135,106957 (2020). https://doi.org/10.1016/j.anucene.2019.106957
[37] Xinxin Mei, Xiaowen Chu, et al. Dissecting GPU memory hierarchy through microbenchmarking. IEEE Tran. Para. Dist. Syst. 28,1 (2016). https://doi.org/10.1109/TPDS.2016.2549523.
[38] B.Hu. Research on application of GPU parallel computing and CUDA programming in environmental engineering. Master dissertation, China University of Geosciences, Beijing (2017).
[39] N. Dai, T. Dai, L.F. Xu et al. Negative flux fixups using positivity-preserving limiters for SN discontinuous finite element scheme of neutron transport equation on unstructured triangular meshes. Anna. Nucl. Energy. 211, 111025 (2025). https://doi.org/10.1016/j.anucene.2024.111025.
[40] P.C.F. Lopes, F. Semeraro, A.M. Pereira et al. Enabling FEM-based absolute permeability estimation in giga-voxel porous media with a single GPU. Comp. Meth. Appl. Mech. Eng. 434, 117559 (2025). https://doi.org/10.1016/j.cma.2024.117559.
[41] Y.L. Zhu, X.W. Chen, C. Hao et al. Implementation of high-fidelity neutronics and thermal–hydraulic coupling calculations in HNET. Nucl. Sci. Tech. 33, 146 (2022) https://doi.org/10.1007/s41365-022-01120-z.
[42] H.Y. Wei, K. Briggs, V. Quintanilla et al. Evaluation of different Krylov subspace methods for simulation of the water faucet problem. Nucl. Sci. Tech. 32,44 (2021) https://doi.org/10.1007/s41365-021-00888-w
[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. Nucl. Sci. Eng. 38,1 (2017). https://doi.org/10.13182/NSE38-8.
[44] W. Xiao, H. Yin, X.J. Liu et al. On the transient models of the VITAS code: applications to the C5G7-TD pin-resolved benchmark problem. Nucl. Sci. Tech. 32,20 (2023) https://doi.org/10.1007/s41365-023-01170-x

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