TYPE Original Research
PUBLISHED 04 January 2023
DOI 10.3389/fnuen.2022.1052332

Hybrid parallel reduction algorithms for the multi-level CMFD acceleration in the neutron transport code PANDAS-MOC

Shunjiang Tao* and Yunlin Xu*

School of Nuclear Engineering, Purdue University, West Lafayette, IN, United States

OPEN ACCESS

EDITED BY
Jun Wang,
University of Wisconsin-Madison,
United States

REVIEWED BY
Jun Sun,
Tsinghua University, China
Han Z,
Tsinghua University, China
Xingfu Wu,
Argonne National Laboratory (DOE),
United States

*CORRESPONDENCE
Shunjiang Tao ,
tao76@purdue.edu
Yunlin Xu,
yunlin@purdue.edu

SPECIALTY SECTION
This article was submitted to Fission and Reactor Design,
a section of the journal
Frontiers in Nuclear Engineering

RECEIVED 23 September 2022
ACCEPTED 05 December 2022
PUBLISHED 04 January 2023

CITATION
Tao S and Xu Y (2023), Hybrid parallel reduction algorithms for the multi-level CMFD acceleration in the neutron transport code PANDAS-MOC.
Front. Nucl. Eng. 1:1052332.
doi: 10.3389/fnuen.2022.1052332

The coarse mesh finite difference (CMFD) technique is considered efficiently in accelerating the convergence of the iterative solutions in the computational intensive 3D whole-core pin-resolved neutron transport simulations. However, its parallel performance in the hybrid MPI/OpenMP parallelism is inadequate, especially when running with larger number of threads. In the original Whole-code OpenMP threading hybrid model (WCP) model of the PANDAS-MOC neutron transport code, the hybrid MPI/OpenMP reduction has been determined as the principal issue that restraining the parallel speedup of the multi-level coarse mesh finite difference solver. In this paper, two advanced reduction algorithms are proposed: Count-Update-Wait reduction and Flag-Save-Update reduction, and their parallel performances are examined by the C5G7 3D core. Regarding the parallel speedup, the Flag-Save-Update reduction has attained better results than the conventional hybrid reduction and Count-Update-Wait reduction.

KEYWORDS

neutron transport, PANDAS-MOC, 2D/1D, multi-level CMFD, reduction, hybrid MPI/OpenMP, parallel computing

1 Introduction

3D whole-core pin-resolved modeling is the state-of-the-art of computational simulation of the neutron transport in the nuclear reactors. Nevertheless, the computational intensiveness makes solving such problems quite challenging, especially for time-dependent transient analysis, where multiple steady-sate eigenvalue and transient fixed-source problems need to be resolved. One popular low-order acceleration scheme for accelerating the convergence of transport solutions is the coarse mesh finite difference (CMFD) method, which was first proposed for nodal diffusion calculation in 1983 (Smith, 1983). It formulates the pinwise core matrices based on the 3D spatial and energy meshes that can be solved simultaneously, and therefore, considerably increases the converge rate of the transport equation.

PANDAS-MOC (Purdue Advanced Neutronics Design and Analysis System with Methods of Characteristics) is a neutron transport code being developed at Purdue University (Tao and Xu, 2022b), and the purpose of which is to provide 3D high-fidelity modeling and simulation of the neutronics analysis in reactor cores. In this code, the 2D/1D method is utilized to estimate the essential parameters to the safety assessment, such as criticality, reactivity, and 3D pin-resolved power distributions, etc. Specifically, the radial solution is determined by the Method of Characteristics (MOC), the axial solution is resolved by the Nodal Expansion Method (NEM), and they are coupled by the transverse leakage. Furthermore, the transport solver is accelerated by the multi-level (ML) CMFD, which is composed by the mutually accelerated multi-group (MG) and one-group (1G) CMFD schemes. However, in the serial computing, the ML-CMFD solver has consumed about 30% of the computation time of the steady state (initial state) calculation in the C5G7 3D test (Tao and Xu, 2022b). Therefore, further acceleration tools shall be considered to improve its efficiency while maintaining the accuracy for the rapid flux changes.

With the considerable increase in the computational power, parallel computing has been widely considered in the high-fidelity neutronics analysis codes, such as DeCART (Joo et al., 2004), MPACT (Larsen et al., 2019), nTRACER (Choi et al., 2018), OpenMOC (Boyd et al., 2016), STREAM (Choi et al., 2021), ARCHER (Zhu et al., 2022) etc., in order to improve the performance of solving the neutron transport equations in the nuclear reactors. Similarly, several parallel models of the PANDAS-MOC have been developed based on the nature of distributed and shared memory architectures, including the Pure MPI parallel model (PMPI), Segment OpenMP threading hybrid model (SGP), and Whole-code OpenMP threading hybrid model (WCP). The detailed design descriptions and parallel performance are presented in Ref (Tao and Xu, 2022a). It is demonstrated that the WCP model costs less memory to finish the calculation, but still needs further improvement in the ML-CMFD solver and the MOC sweep solver in order to attain comparable, even exceptional, performance to the PMPI code in order to show the benefits of the hybrid memory architectures. The improvement of the MOC sweep solver is discussed in Ref. (Tao and Xu, 2022c), and this article will concentrate on the optimization of the ML-CMFD solver.

Additionally, while executing large amounts of processors in hybrid MPI/OpenMP cases, the parallel performance of CMFD could be quite inadequate (Tao and Xu, 2020), which is also true in the WCP model of PANDAS-MOC. In Ref (Tao and Xu, 2022a), it is found that one major challenge of the hybrid parallel of ML-CMFD is the reduction operation, which is the principal problem we are trying to resolve in this work. In this paper, we will firstly discuss the conventional hybrid OpenMP/MPI reduction pattern, which is also utilized in the WCP code in the first place. Then two innovative reduction algorithms will be proposed: Count-Update-Wait reduction and Flag-Save-Update reduction. Next, the parallel performance for the ML-CMFD solver implementing those three reduction methods are measured and contrasted with each other, which is expected to demonstrate the breakthrough of the newly designed reduction methods. Last, the performance of the optimized WCP mode will be briefly discussed to show the overall improvement from the novel hybrid reduction algorithm and MOC parallelization schemes.

2 Methodology

2.1 PANDAS-MOC methodology

This section will briefly introduce the methodology of the PANDAS-MOC, highlighting the concepts that are most relevant for this work. The detailed derivations can be found in Ref. (Tao and Xu, 2022b). The transient method starts with the 3D time-dependent neutron transport equation Eq. 1 and the precursor equations Eq. 2:

$$\frac{1}{v_g(r)}\frac{\partial \varphi_g(r,\Omega,t)}{\partial t} = -\Omega\cdot\nabla\varphi_g(r,\Omega,t) - \Sigma_{tg}(r,t)\varphi_g(r,\Omega,t) + S_{sg}(r,\Omega,t) + \frac{\chi_g(r)}{4\pi}S_F(r,t) + \frac{1}{4\pi}\sum_k \chi_{dgk}\left(\lambda_k C_k(r,t) - \beta_k S_F(r,t)\right) \tag{1}$$

$$\frac{\partial C_k(r,t)}{\partial t} = \beta_k(r)S_F(r,t) - \lambda_k(r)C_k(r,t),\quad k = 1,2,\ldots,6 \tag{2}$$

The terms χ_g, S_{sg} and S_F are the average fission neutron spectrum, scattered neutron source, and prompt fission neutron source, and they are defined as Eq. 3, Eq. 4, and Eq. 5 respectively. The definitions of the rest variables are summarized in Table 1.

$$\chi_g = \chi_{pg} + \sum_k \beta_k\left(\chi_{dgk} - \chi_{pg}\right) \tag{3}$$

$$S_{sg}(r,\Omega,t) = \sum_{g'}\int_{4\pi}\Sigma_{g'g}(r,\Omega'\cdot\Omega,t)\varphi_{g'}(r,\Omega',t)d\Omega' \tag{4}$$

$$S_F(r,t) = \frac{1}{k^s_{eff}}\sum_{g'}\nu\Sigma_{fg'}(r,t)\int_{4\pi}\varphi_{g'}(r,\Omega,t)d\Omega \tag{5}$$

To numerically solve the transport equation, several approximations are considered:

1. Angular flux: Exponential transformation
2. Time derivative term: Implicit scheme of temporal integration method
3. Fission source: Exponential transformation and linear change in each time step

TABLE 1 Definitions of terms in the time dependent transport equation.

Term	Definition	Term	Definition
φ	Angular flux	k	Precursor group index
g	Energy group index	β_k	Delayed neutron fraction
Σ_{tg}	Total macroscopic cross-section	C_k	Density of delayed neutron precursors
$\Sigma_{g'g}$	Scatter cross-section	λ_k	Decay constant
ν	Prompt fission neutron yield	χ_{dgk}	Delayed fission neutron spectrum
Σ_{fg}	Fission cross-section	χ_{pg}	Prompt fission neutron spectrum
k_{eff}^{s}	Eigenvalue of the initial state		

4. Densities of delayed neutron precursors: integrating Eq.2 over time step

Accordingly, Eq. 1 can be transformed to the Transient Fix Source Equation, and the Cartesian form of which is:

$$\left(\eta\frac{\partial}{\partial x}+\epsilon\frac{\partial}{\partial y}+\mu\frac{\partial}{\partial z}\right)\varphi_g^n(r,\Omega)+\Sigma_{tg}^n(r)\varphi_g^n(r,\Omega)$$
$$=S_{sg}^n(r,\Omega)+\frac{1}{4\pi}\left[\chi_g(r)S_F^n(r)+S_{ntg}^n(r)+S_{trg}^{n-1}(r)\right] \tag{6}$$

where

$$r=(x,y,z),\quad \Omega=(\mu,\alpha),\quad \eta=\sin\theta\cos\alpha,\quad \epsilon=\sin\theta\sin\alpha$$
$$S_{trg}^{n-1}(r)=S_{dcg}^{n-1}(r)+S_{dtg}^{n-1}(r),\qquad S_{dcg}^{n-1}(r)=\sum_k\chi_{dkg}\lambda_k\hat{C}_k^{n-1}(r)$$
$$S_{dtg}^{n-1}(r)=\frac{\phi_g^{n-1}(r)}{E_g^n(r)\Delta t_n v_g(r)},\qquad S_{ntg}^n(r)=\hat{\chi}_g S_F^n(r)-\Sigma_{dg}^n\phi_g^n$$
$$E_g^n(r)=e^{-\alpha_g^n(r)\Delta t_n},\qquad \alpha_g^n=\left(\log P_n^{tot}-\log P_{n-1}^{tot}\right)\Big/\Delta_{n-1},$$
$$\Sigma_{dg}^n=\frac{\alpha_g^n}{v_g}+\frac{1}{\Delta t_n v_g}$$

Instead of directly resolve the computational-intensive 3D problem, it is converted to a radial 2D problem and an axial 1D problem, which is conventionally referred to as the 2D/1D method. The 2D equation is obtained by integrating Eq. 6 axially over the axial plane ($\int_{z_b}^{z_t}dz$) and solved by the MOC method, and the 1D equation is obtained by radially integrating over a box ($\int\int_A dxdy$) and solved by the NEM method. Then the 2D radial solution and 1D axial solution are coupled by the transverse leakage. Besides, the ML-CMFD approach is implemented to accelerate the convergence for solutions to the Transient Fix Source Equation. Since this paper concentrates on the performance improvement on the ML-CMFD solver, the details of MOC and NEM are omitted for brevity and the details could be found in Ref (Xu and Downar, 2012) (Tao and Xu, 2022b) (Tao and Xu, 2022c).

2.2 Multi-level CMFD method

The transport equation is generally very hard to be solved for large reactor cores. But the finite difference equation for neutron diffusion equation can be solved very efficiently for nuclear reactor (Hao et al., 2018) (Tao and Xu, 2020). In the PANDAS-MOC implementation, the CMFD is implemented by overlaying a 2D Cartesian mesh over the FSR meshes. The coarse mesh layout used for solving a 17 × 17 PWR assembly problem is illustrated in Figure 1B, where each colored cell denotes a different region. While performing MOC sweeping, each CMFD region could be further discretized to multiple flat source regions, as shown in Figure 1C. The bridges connecting two mesh layouts are the variable homogenization and current coupling coefficients.

The MG-CMFD formulations are obtained from integrating the transient transport equation over a 4π and over the node n of interest. The equation for steady state is listed in Eq. 7 and for transient analysis is written in Eq.9. In addition, the 1G-CMFD equations are determined *via* collapsing the energy dimension from the MG-CMFD formula, and Eq.8 and Eq.10 are the steady state and transient equation respectively.

$$\frac{1}{V^c}\sum_s A_s^c\left(\tilde{\tilde{D}}_{g,s}^c\bar{\phi}_g^n-\tilde{\tilde{D}}_{g,c}^s\bar{\phi}_g^{n,s}\right)+\Sigma_{rg}^n\phi_g^n-\sum_{g'\neq g}\Sigma_{gg'}^n\phi_{g'}^n=\lambda^{n-1}\chi_g S_F^{n-1} \tag{7}$$

$$\frac{1}{V^c}\sum_s A_s^c\left(\tilde{\tilde{D}}_{A,s}^c\bar{\phi}_A^n-\tilde{\tilde{D}}_{A,c}^s\bar{\phi}_A^{n,s}\right)+\left(\Sigma_{rA}^n-\lambda_s\nu\Sigma_{fA}^n\right)\phi_A^n$$
$$=\left(\lambda^{n-1}-\lambda_s\right)S_F^{n-1} \tag{8}$$

$$\frac{1}{V^c}\sum_s A_s^c\left(\tilde{\tilde{D}}_{g,s}^c\bar{\phi}_g^n-\tilde{\tilde{D}}_{g,c}^s\bar{\phi}_g^{n,s}\right)+\tilde{\Sigma}_{rg}^n\phi_g^n-\sum_{g'\neq g}\Sigma_{gg'}^n\phi_{g'}^n$$
$$=\left(\chi_g+\hat{\chi}_g\right)S_F^n+S_{dcg}^n+S_{dtg}^n \tag{9}$$

$$\frac{1}{V^c}\sum_s A_s^c\left(\tilde{\tilde{D}}_{A,s}^c\bar{\phi}_A^n-\tilde{\tilde{D}}_{A,c}^s\bar{\phi}_A^{n,s}\right)$$
$$+\left\{\tilde{\Sigma}_{rA}^n-\left[1+\sum_k\lambda_k^n\Omega_k^n-\beta\right]\lambda\nu\Sigma_{fA}^n\right\}\phi_A^n$$
$$=S_{dcA}^n+S_{dtA}^n \tag{10}$$

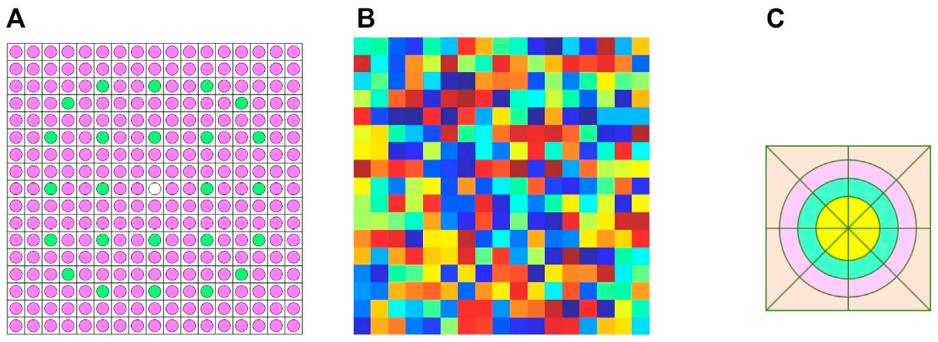

FIGURE 1
CMFD and FSR mesh layout for a 17 × 17 PWR assembly. **(A)** PWR assembly **(B)** CMFD mesh layout **(C)** FSR layout in one CMFD mesh.

In the ML-CMFD technique, the multi-group solution and one-group solution are employed to mutually speed up the convergence. Once the multi-group node average flux and the surface current are ready, the one-group parameters can then be updated accordingly. And this new 1G-CMFD linear system can render next-generation of one-group flux distribution which will be used to update the multi-group flux parameters to benefit the eigenvalue convergence. The flowchart of its implementation in the PANDAS-MOC is illustrated in Figure 2. Moreover, the ML-CMFD scheme used for the steady state case and the transient case are approximately the same, except that there is no update of the eigenvalue for the transient evaluation. Therefore, the steady state module terminates when the change of the eigenvalue and the norm of the difference of the MOC flux between two consecutive iterations have both satisfied the convergence criteria, while the transient iteration stops when the norm of the MOC flux difference is satisfied. More details can be found in Ref (Tao and Xu, 2022b).

Meanwhile, the CMFD Eqs 7–10 are large sparse asymmetric linear systems, and can be rearranged to classic matrix notation, $Ax = b$, where A is a block-diagonal sparse matrix corresponding to the coefficients and cross-section parameters whose dimension is about hundreds of millions with respect to the large reactor size, and b represents the source term. To further accelerate the convergence of the solution to this sparse linear system, the preconditioned generalized minimal residual method (GMRES) is employed because it can sufficiently take advantage of the sparsity of the matrix and is easy to implement (Saad, 2003). In addition, Givens rotation is applied in the GMRES solver, since it can preserve better orthogonality and is more efficient than other processes for the QR factorization of the Hessenberg matrix.

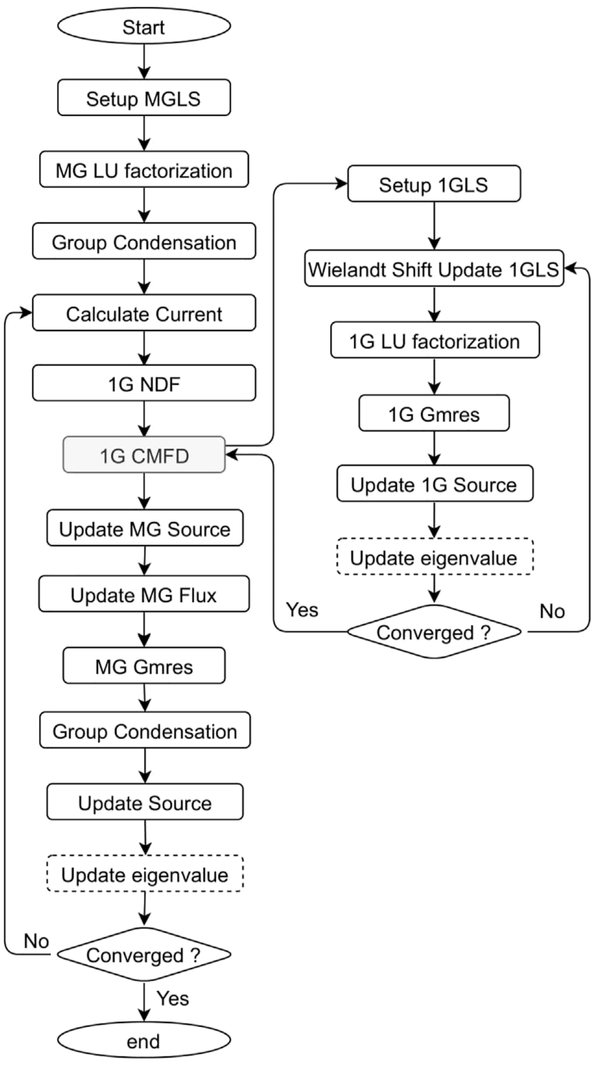

FIGURE 2
SS/TS Multi-level CMFD (dashed step is only valid for SS).

2.3 Hybrid parallelism of ML-CMFD

Generally speaking, there are two ways to convert the serial program to multithreading form for large problems:

MPI-only and hybrid MPI/OpenMP. In many cases, the hybrid programs execute faster than the pure MPI program because they have lower communication overhead (Quinn, 2003). Suppose the program is executed on a cluster of *a* multiprocessors and each has *b* CPUs. The program using MPI must create *ab* processors to use every CPU, thus all *ab* processors are involved in the communication. On the contrary, the program using hybrid MPI/OpenMP only need to create *a* processors, and each processor then spawns *b* threads to divide the workload in the parallel sections, which also has employed every CPU. Nevertheless, only that *a* processors are active during the communication procedure, which will give the hybrid program lower communication overhead than the MPI program. In this paper, all work are performed based on the previous developed whole-code OpenMP threading hybrid model (WCP) of the PANDAS-MOC, which partitioned the entire workload by the MPI and OpenMP simultaneously in order to get rid of the repetitious creation and suspension of the OpenMP parallel regions. The detailed information on the WCP model could be found in Ref (Tao and Xu, 2022a).

As indicated by the CMFD Eqs 7–10, the principal target of the ML-CMFD is to find the solution x to the linear system $Ax = b$ by the GMRES iterative scheme. Therefore, the major work included in this part is the matrix and vector operations, such as the vector-vector dot product, matrix-vector multiplication, and matrix-matrix multiplication. Take vector-vector dot product as an example, the general way to compute dot-product using hybrid MPI/OpenMP scheme is listed in Algorithm 1, where nx, ny, nz are the spatial dimensions and ng is the neutron energy group dimension. In order to find the dot-product result, three steps are involved: 1) each OpenMP thread compute the partial result on their own, 2) a local result will be collected for each MPI processor from its launched OpenMP threads by the omp reduction clause, 3) the global result is computed and broadcasted by the MPI_Allreduce() routine. This reduction conception is also implemented in the WCP in the first place, and its performance will be discussed in the following Section 4 in details.

```
#pragma omp for reduction(+: local_dot) collapse(4)        ▷ threadprivate: k, j, i, g, local_dot
for $k = 0, 1, 2, .., nz - 1$ do
    for $j = 0, 1, 2, .., ny - 1$ do
        for $i = 0, 1, 2, .., nx - 1$ do
            for $g = 0, 1, 2, .., ng - 1$ do
                local_dot += (v1[k][j][i][g] * v2[k][j][i][g])
#pragma omp single
MPI_Allreduce(&local_dot, &global_dot, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

Algorithm 1. Hybrid MPI/OpenMP Dot-product Algorithm.

2.4 Performance metrics

The measurement of parallel performance is the speedup, which is defined as the ratio of the sequential runtime (T_s) which is approximated using the runtime with 1 processor (T_1) in pure MPI model, and the parallel runtime while utilizing p processors (T_p) to solve the same problem. Also, efficiency (ϵ) is a metric of the utilization of the resources of the parallel system, the value of which is typically between 0 and 1.

$$S_p = \frac{T_s}{T_p} \approx \frac{T_1}{T_p}, \quad \epsilon_p = \frac{S_p}{p} \tag{11}$$

3 Test problem

The parallel performance of designed codes in this work are determined by a steady state problem, in which all control rods are removed from the C5G7 3D core from the OECD/NEA deterministic time-Dependent neutron transport benchmark, which is proposed to verify the ability and performance of the neutronics codes without neutron cross-sections spatial homogenization above the fuel pin level (Boyarinov et al., 2016) (Hou et al., 2017). It is a miniature light water reactor with 8 uranium oxide (UO_2) assemblies, 8 mixed oxide (MOX) assemblies, and surrounding water moderator/reflector. Besides, the C5G7 3D model is a quarter-core and fuel assemblies are arranged in the top-left corner. For the sake of symmetry, reflected condition is used for the north and west boundaries, and vacuum condition is considered for the rest six surfaces. Figure 3A is the planar and axial configuration of the C5G7 core. The size of the 3D core is 64.26 $cm \times$ 64.26 $cm \times$ 171.36 cm, and the axial thickness is equally divided into 32 layers.

Moreover, the UO_2 assemblies and MOX assemblies have the same geometry configurations. The assembly size is 21.42 $cm \times$ 21.42 cm. There are 289 pin cells in each assembly arranged as a 17 × 17 square (Figure 3B), including 264 fuel pins, 24 guide tubes, and 1 instrument tube for a fission chamber in the center of the assembly. The UO_2 assemblies contains only UO_2 fuel, while the MOX assemblies includes MOX fuels with 3 levels of enrichment: 4.3%, 7.0%, and 8.7%. Besides, each pin is simplified as 2 zones in this benchmark. Zone 1 is the homogenized fuel pin from the fuel, gap, cladding materials, and zone 2 is the outside moderator (Figure 3C). The pin (zone 1) radius is 0.54 cm, and the pin pitch is 1.26 cm.

While running the 3D benchmark problem, the essential parameters are defined as following. For the MOC sweeping, all cases were performed with Tabuchi-Yamamoto quadrature set with 64 azimuthal angles, 3 polar angles, the ray-spacing was 0.03 cm. The fuel pin-cells are discretized into 8 azimuthal flat source regions and 3 radial rings (Figure 1C), and the moderator cells were divided into 1 by 1 or 6 by 6 coarse meshes, which depends on their locations in the core. Meanwhile, for the numerical iterative functions, the convergence criteria of the GMRES method in the ML-CMFD solver was set as 10^{-10}, and the convergence criteria for the eigenvalue was 10^{-6} and for the flux

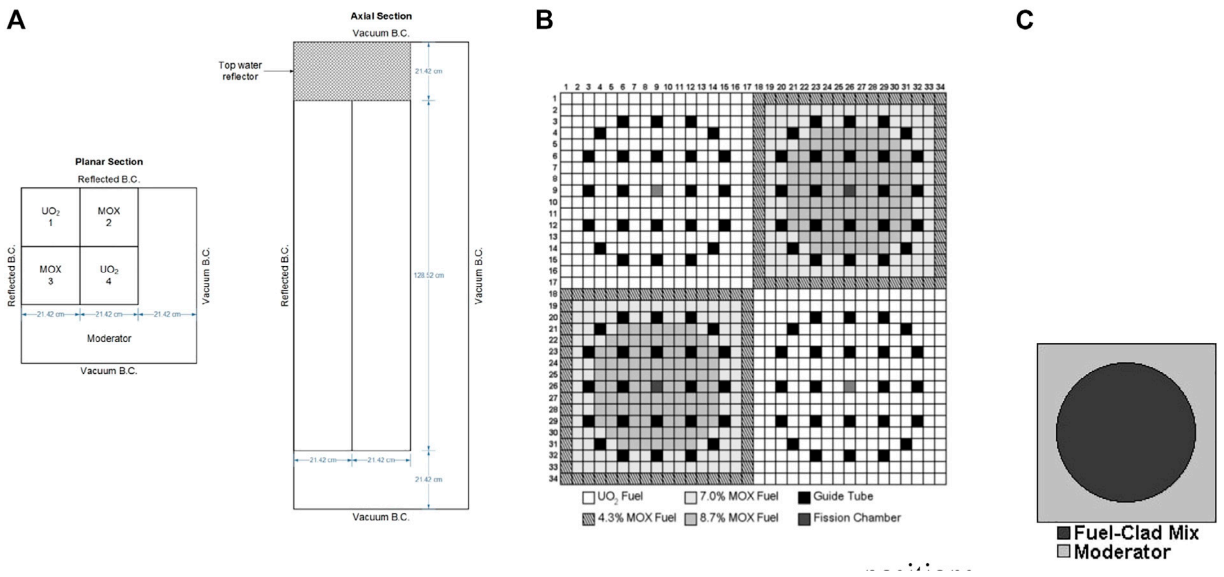

FIGURE 3
Geometry and composition of C5G7-TD benchmark. **(A)** Planar and axial view of 1/4 core map **(B)** Fuel pin compositions and number scheme **(C)** Pin cell.

was 10^{-5}. No preconditioners were employed to show the improvement from the algorithm update.

This study is conducted in the "Current" cluster at Purdue University, the mode of which is Intel(R) Xeon(R) Gold 6152 CPU @ 2.10GHz, and it has 2 NUMA nodes, 22 cores for each node, and 1 thread per core. The parallel performance is measured using Intel(R) MPI Library 2018 Update 3, and the compiler is mpigcc (gcc version 4.8.5).

4 Parallelism of reduction

According to the numerical experiments in Ref. (Tao and Xu, 2022a), the predominant factors that hurt the hybrid parallel performance of the ML-CMFD solver are the repetitious construct and destruct of the OpenMP parallel region and the reduction procedure. The former has been tackled in the WCP design, which has moderately improved the speedup of the ML-CMFD solver. This section will first discuss the difficulties of the conventional method of the hybrid MPI/OpenMP reduction, and then propose two novel reduction designs to further advance the efficiency of the parallelism in the ML-CMFD solver.

4.1 Hybrid default reduction

The majority work in the multi-level CMFD solver are the matrix-vector, matrix-matrix, and vector-vector product, which means that summing the partial results given by each thread is playing a vital role to the computation accuracy and efficiency. In general, such reduction process in the hybrid MPI/OpenMP program is finished by the omp reduction clause from OpenMP library in the first place and then by the MPI_Allreduce() routine from MPI library (Figure 4A), which is referred as "hybrid default reduction" in the context hereafter. The shortcoming of this reduction manner is that the omp reduction only conducts the data synchronization when all spawned threads have finished their tasks due to the implicit barrier, which may slow down the calculation. Figure 4B is an example of the hybrid default reduction. If thread 0 need $t_{p,0}$ time to finish its task, yet thread 2 only need $t_{p,2}$ time, then thread 2 will be staying idle and wait for $t_{w,2}$ time before the OpenMP reduction can be executed. Of course, this will happen again on the MPI processors before the MPI_Allreduction() operation can be performed. In addition, if "nowait" is specified to override such implicit barriers simultaneously with the omp reduction clause, it will cause race condition and unpredicted behaviors. To overcome the impacts caused by the implicit barrier after the OpenMP reduction and reduce the thread idle time, here we designed two different reduction algorithms.

4.2 Count-update-wait reduction

Instead of waiting for all OpenMP threads having their partial results ready, this method immediately gathers the partial result once a thread has finished its calculation and

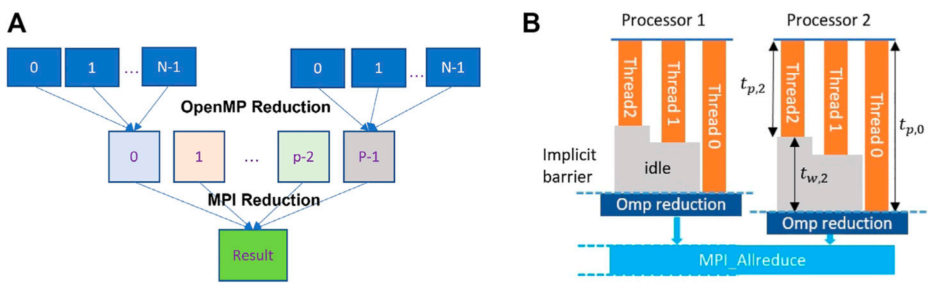

FIGURE 4
Example of hybrid default reduction. **(A)** Hybrid MPI + OpenMP reduction **(B)** Reduction time.

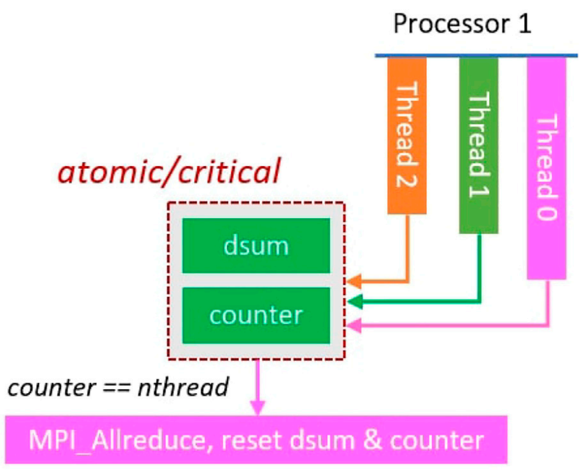

FIGURE 5
Illustration of count-update-wait reduction.

uses a global counter to count the number of collected data. Till all threads have provided their partial results, in the zeroth OpenMP thread, the MPI_Allreduction() routine is utilized to compute the global result and reset the counter and variables. For example, to compute the hybrid reduction sum, the defined function is listed in Algorithm 2 and illustrated in Figure 5, in which the counting and summation processes are performed atomically to take advantage of the hardware provided atomic increment operation. Furthermore, the Count-Update-Wait reduction functions for finding the global maximum value, global minimum value, or any combinations of those operations were written in the similar fashion, except that the counting and local result updating operations are protected by the omp critical directive if necessary.

In the implementation, the omp reduction clause is removed from the OpenMP #pragma statement and the nowait clause is applied to override the implicit OpenMP barrier. In that regard, the Count-Update-Wait algorithm has one barrier less than the default reduction method. However, the trade-offs are the incurred synchronization overhead every time a thread enters and exits the atomic or critical section and the inherent cost of serialization.

```
Initialize global variables: counter, local_sum
function REDUCTION(psum, tid, nthreads)                       ▷ tid=omp_get_thread_num()
    #pragma omp atomic
    counter++                          ▷ counting the number of OpenMP threads have arrived
    #pragma omp atomic
    local_sum += psum                  ▷ collecting the partial sum from the arrived thread
    if tid == 0 then
        while counter < nthreads do
            waiting
        MPI_Allreduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
        reset local_sum, counter
    #pragma omp barrier                ▷ Assuring MPI_Allreduce is done before any thread return
    return global_sum
```

Algorithm 2. Count-Update-Wait Reduction Algorithm.

4.3 Flag-save-update reduction algorithm

In the previous Count-Update-Wait Reduction algorithm, the atomic and critical clauses, which are utilized to guarantee the race-free conditions, introduce synchronization overhead and serialization cost and hence decrease the computation efficiency. Since the purposes of the atomic and critical clauses are restricting a specific memory location or a structured block to be accessed by a single thread at a time, which otherwise is considered as a race condition, they can be eliminated from the code if the partial results are saved in different memory locations before the final reduction.

In the Flag-Save-Update Reduction, the MPI reduction is still performed by the MPI_Allreduction() routine, and the improvement is carried out on the OpenMP reduction procedure.

Firstly, the OpenMP threads are deliberately configured as a tree structure. In such sense, this procedure allows us to collect the data in $log_2(N_{threads})$ stages, rather than $N_{threads}-1$ stages, which can be a huge saving in reduction. The

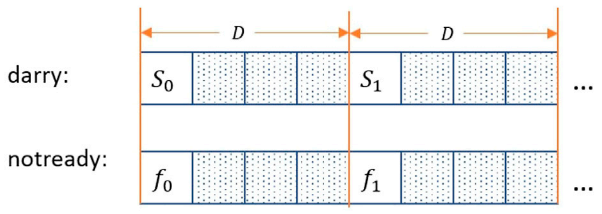

FIGURE 6
Arrays structure.

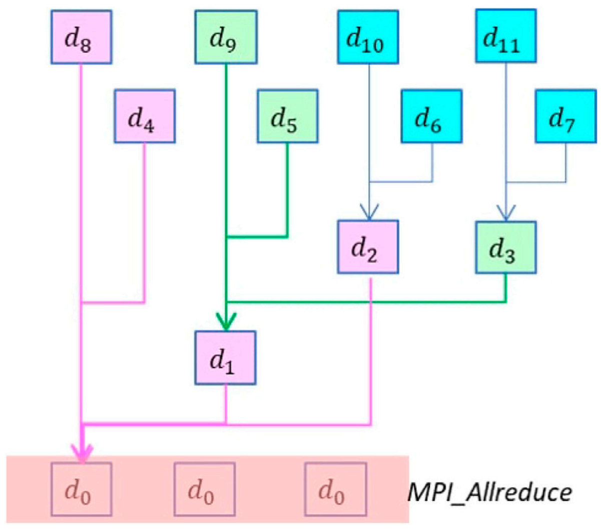

FIGURE 7
Parent-child layout of OpenMP threads (total threads = 12).

maximum number of children to each thread is determined as:

$$N_{\text{children}} = \lfloor \log_2 N_{threads} \rfloor \tag{12}$$

Next, two global arrays are allocated to store the partial results and the status flag of each OpenMP thread, *darray* and *notready*, and their sizes are defined as $N_{threads}{}^{*}D$, where D is the spacing distance that isolates the element from each OpenMP thread and prevents the potential false sharing and race condition problems. Once the thread partial result is evaluated, it will be stored to the corresponding location in the *darray* and the flag in the *notready* will be updated, as showing in Figure 6. Since all locations are separated from each other, reading and modifying their values will not incur any race condition issues for the appropriate D value. We are going to use $D = 128$ in this work to measure the performance hereafter, which is chosen from sample experimentation.

Furthermore, in this method, a global status flag and a thread-private status flag are designed to control the calculation flow in order to make sure that the collected data are fresh and to avoid issues like missing data or recurrently reading the same data.

The Flag-Save-Update reduction algorithm is designed according to the executed number of OpenMP threads and MPI processors and listed in Algorithm 3. Figure 7 is an illustration of the thread scheme when the total OpenMP thread is 12, in which arrows with different colors indicate different computation stages. To perform the OpenMP reduction, threads having a different number of children threads are defined behaving differently:

1. If a thread has no children, then flip its thread-private flag, save its result to *darray*, and update status in the *notready* array
2. If a thread has children, for example, when d_1, d_3, d_5, and d_9 are ready, their results are collected and then store to d_1 location in *darray*, and update status at the corresponding locations in the *notready* as well.
3. All partial results should be collected to d_0 after $log_2(N_{threads})$ stages of reduction.

Furthermore, MPI_Allreduce() is called at thread 0 to perform the mpi reduction if it is necessary, and the global flag is flipped to allow all threads to access and read the final result.

With respect to its implementation in the PANDAS-MOC code, the reduction clause is removed from the OpenMP pragmas, and the "nowait" clause is specified to override the implicit barrier after the OpenMP regions. Therefore, it contains neither implicit nor explicit barriers in the work flow, comparing to the hybrid default reduction and the Count-Update-Wait reduction.

```
Initialize global arrays: darray, notready
Initialize flags: even, working_on_even                          ▷ even: threadprivate
Establish the thread tree structure                    ▷ threadprivate array: children
if $tid > 0$ then                        ▷ collecting partial results from children threads
    if $N_{\text{children}} == 0$ then
        flip thread private flag
        Save the partial result to the corresponding location in darray
        Flip the status flag to the corresponding location in notready
        Wait until even == working_on_even
    else if $N_{\text{children}} == 1$ then
        flip thread private flag
        Wait until the child thread is ready
        sum of its partial result and value at child location in darray
        Save the sum to the corresponding location in darray
        Flip the status flag to the corresponding location in notready
        Wait until even == working_on_even
    else
        flip thread private flag
        Wait until the child thread is ready
        sum of its partial result and all values at children location in darray
        Save the sum to the corresponding location in darray
        Flip the status flag to the corresponding location in notready
        Wait until even == working_on_even
if $N_{\text{MPI}} == 1$ then
    sum up results from all children threads
    flip the global flag
    return result
else                                                              ▷ hybrid reduction
    sum up results from all children threads
    collect the results from all processors: MPI_Allreduce() at tid 0
    flip the global flag to allow the threads to read the final result
    return result
```

Algorithm 3. Flag-Save-Update MPI/OpenMP Reduction Algorithm.

4.4 Performance

In order to compare the hybrid parallel performance, the total executed number of threads were fixed as 36. Given that the C5G7 3D core has 32 layers in the axial direction, the number of MPI processors applied in the axial direction are the common factors of 32 and 36 to achieve better load balance. All combinations of number of MPI processors and OpenMP threads and the x–y–z distribution of MPI processors are tabulated in Table 2. Besides, all tests were executed 5 times to minimize the measurement error and their average run-time were used for the further data analysis.

The measured speedup and efficiency for the ML-CMFD solver using the hybrid default reduction, Count-Update-Wait reduction, and Flag-Save-Update reduction are computed based on the run-time for the ML-CMFD solver from the pure MPI model (PMPI) with single MPI processor (2196.113 s) (Tao and Xu, 2022a), and they are presented in Figure 8, which presents big improvement from the default reduction to the Flag-Save-Update reduction, especially when testing with large number of OpenMP threads.

All hybrid reductions have their largest speedup at (36, 1). Although theoretically results are expected similar to each other when running with a single thread, the actual measured speedup shows that Flag-Save-Update reduction > hybrid default reduction > Count-Update-Wait reduction, and they all present sublinear speedup. In this test, the inherent overhead of OpenMP is unavoidable, such as the process of generating multi-threading and walking through the included OpenMP directives even though there was only a single thread spawned by each MPI processor. Among three hybrid methods, the Flag-Save-Update reduction rendered the largest speedup because the synchronization points have been deliberately eliminated as many as possible from the algorithm. Besides, even though the Count-Update-Wait reduction had one implicit barrier less than the hybrid default reduction, it still had slightly larger natural overhead due to the existence of the atomic or critical structure. In the hybrid default reduction, the implicit barrier introduced by the "reduction" clause forces all threads to wait until all threads in the same team have arrived to this place and then gathers their partial results. In the Count-Update-Wait reduction, its counterpart is the atomic/critical structure, which collects the partial results from the threads immediately when they are arrived (Algorithm 2). When running with single thread, the implicit barrier in the hybrid default reduction could hardly slow down the calculation, whereas the serialization cost incurred by the atomic/critical structure not negligible. The measured difference between them indicates that the serialization cost incurred by the atomic or critical is more expensive than the implicit barrier in this circumstance.

The observed trend of speedup from Count-Update-Wait reduction and from hybrid default reduction came cross at (4,9). When the number of executed OpenMP threads is smaller than 9, the speedup accomplished by the Count-Update-Wait reduction is about 96%–97% of the hybrid default reduction. This is because the workload assigned to each OpenMP thread is not significantly biased, the waiting to completion time are trivial in both algorithms. Yet, the overhead brought by the atomic and critical serializations for collecting the OpenMP partial results and updating the counters will introduce extra cost and slow down the calculation. However, as more OpenMP threads contributed, the benefit of owning fewer synchronization barriers has been demonstrated as the Count-Update-Wait reduction reached larger speedup. After compensating the inherent cost of serialization because of omp atomic and critical blocks, its speedup at (1, 36) is about 30.9% higher than the hybrid default reduction.

On the other hand, the Flag-Save-Update reduction has achieved larger speedup than the other two algorithms for all tests due to lacking of synchronization points, even when there is only a single OpenMP contributing to the calculation [i.e., (36,1)]. Moreover, this advantage was more obvious when larger number of OpenMP threads were executed. In Figure 8, the speedup ratio of the Flag-Save-Update reduction to the hybrid default reduction has augmented from 1.05 to 1.80 from right to left as more OpenMP contributed and fewer MPI devoted. Concerning the Flag-Save-Update reduction, the obtained speedup of most tests falls within the range of (17, 19). However, The speedup could also be weakened by the communications and data synchronizations among MPI processors if there were domain decomposition in the x-y plane, for instance the drop in (12,3). Despite that, the Flag-Save-Update reduction can compensate part of this overhead, since its ratio of speedup at (12,3) and at (4,9) was larger than the other two reduction methods. Furthermore, overall tests, (1, 36) has attained the smallest speedup. Other than the OpenMP thread load balance among and inherent overhead from the algorithm *per se*, the parallel performance of this group could possibly be restricted by the hardware properties, such as memory bandwidth, cache size and cache hit rate. For example, the size of *darray*, the type of elements contained in which is double, is about $N_{threads} \times D \times 8/1024\,kb = 36 \times 128 \times 8/1024\,kb = 36\,kb$, however the size of the Ld1 and Ldi cache of the "Current" cluster are 32 kb.

5 WCP optimizations

As explained in the Ref (Tao and Xu, 2022a), the parallel performance achieved from the original WCP model is better than the SGP model which is the conventional fashion of hybrid MPI/OpenMP parallelization, yet not comparable to the PMPI model. The advancement of the hybrid reduction in ML-CMFD solver and parallelism of MOC sweeping are desired to optimize the WCP code and make it at least comparable to the PMPI code. In this article, we have proposed several innovative algorithms to conduct the hybrid reduction, and the most promising one is the Flag-Save-Update reduction. Its collected performance is better than the hybrid default reduction, regardless that it is still slower than the MPI_Allreduction() standalone in the PMPI mode while using the identical total number of threads (Figures 8, 9A). On the other hand, the improvement on the parallelism of MOC sweeping is

TABLE 2 Combinations of tested number of MPI and OpenMP threads.

Total threads	MPI	OpenMP	MPI-(x, y, z)	Total threads	MPI	OpenMP	MPI-(x, y, z)
36	1	36	(1,1,1)	36	6	6	(3,1,2)
36	2	18	(1,1,2)	36	12	3	(3,1,4)
36	4	9	(1,1,4)	36	18	2	(3,3,2)
				36	36	1	(3,3,4)

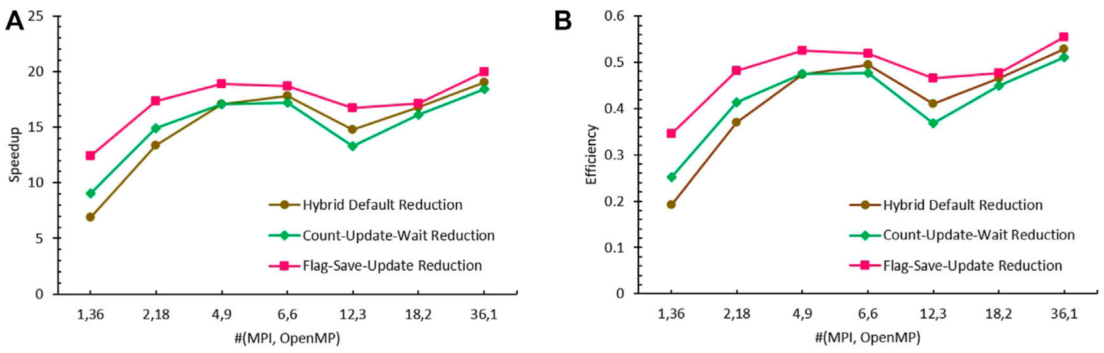

FIGURE 8
Comparison of ML-CMFD solver performance using different reduction algorithms. **(A)** Speedup **(B)** Efficiency.

thoroughly discussed in Ref (Tao and Xu, 2022c) and here is a brief description.

While using the OpenMP directives to partition the characteristic rays sweeping, it is found that the speedup of the MOC sweep are limited by the unbalanced workload among threads and the serialization overhead caused by the omp atomic clause for assuring the correctness of MOC angular flux and current update (Tao and Xu, 2022a). Two schedules are designed to resolve these two obstacles: Equal Segment (SEG) schedule and No-Atomic schedule. The SEG resolves the unbalanced issue, in which the average number of segments are determined in the first step according to the total number of segments and number of executed threads, and then long-tracks are partitioned based on this average number so that the number of segments (i.e., actual computational workload) are distributed among OpenMP threads as equalized as possible. Based on SEG, the No-Atomic schedule further removes all omp atomic structures to improve the computational efficiency by pre-defining the sweeping sequence of long-track batches across all threads to create a race-free job. The implementation details could be found in Ref. (Tao and Xu, 2022c). After repeating the same tests on each schedule, it is confirmed that the No-Atomic schedule is capable to achieve much better performance than the other schedules, and even outperforms the PMPI sweeping with the identical total number of threads.

The comparison of the parallel performance obtained from the PMPI (dashed blue line), the original WCP using default methods [green line, labeled as (d)], and the optimized WCP with Flag-Save-Update reduction and No-Atomic schedule [pink line, labeled as (o)] are illustrated in Figure 9. Although the speedup of the ML-CMFD solver using the Flag-Save-Update reduction is still slower than the PMPI using MPI_Allreduction() (Figure 9A), the enhancement accomplished from the optimized No-Atomic MOC sweep is large enough to compensate such weakness (Figure 9B). Therefore, pertaining to the overall steady state calculation, the optimized WCP code has managed to obtain much better speedup than the original WCP code, and comparable to or even greater than the PMPI code as explained in Figures 9C,D.

6 Summary remarks

The CMFD technique is commonly used in the neutron transport simulations to accelerate the convergence of the iterative solutions. However, the performance of its implementation in the hybrid MPI/OpenMP parallelism is inadequate, especially when running with larger number of threads. In the original WCP model of the PANDAS-MOC code, the hybrid MPI/OpenMP reduction has been determined as the principal issue that restraining the parallel speedup of the ML-CMFD solver. Conventionally, the hybrid reduction operation is finished by exploiting the omp reduction clause and MPI_ Allreduction() routine, which was referred to as “hybrid default reduction” in the previous context. Other than that, two original

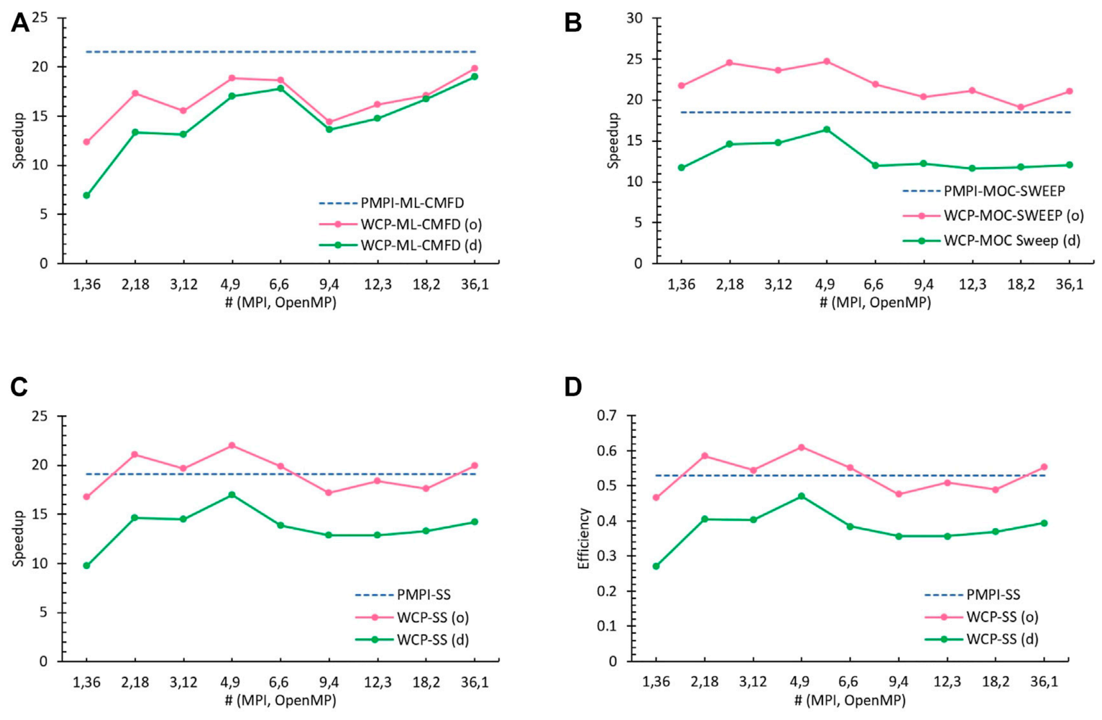

FIGURE 9
Performance Comparison with total threads as 36 (o: optimized algorithm, d: default method). **(A)** Speedup: ML-CMFD (d: hybrid default reduction, o: Flag-Save-Update reduction) **(B)** Speedup: MOC sweep (d: long-track schedule (default), o: No-Atomic schedule) **(C)** Speedup: Steady State **(D)** Efficiency: Steady State.

hybrid reduction algorithms were presented in this paper: Count-Update-Wait reduction and Flag-Save-Update reduction. The first algorithm had fewer barriers than the hybrid default reduction yet introduced extra synchronization points, such as atomic or critical sections, to count and assure that all partial results were collected from the launched OpenMP threads. Besides, the second algorithm was accomplished by using the global arrays and status flags, and establishing the tree configuration of all threads, where includes no implicit and explicit barriers. The hybrid parallel performance for three algorithms were examined using the C5G7 3D core and the total number of threads were set as 36. The Flag-Save-Update reduction yielded the best speedup at all tested cases, and its superiority was more obvious as more OpenMP threads were contributed. For instance, when using (1, 36), the obtained speedup of Flag-Save-Update reduction algorithm is about 1.8 times of the speedup achieved by the hybrid default reduction. In spite of the fact that it is less efficient than the MPI_Allreduction() routine in PMPI model, this disadvantage can be offset by the great advance from the optimized MOC sweep scheme, and together they are able to reach larger speedup than the PMPI model when using identical total number of threads.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding authors.

Author contributions

All authors contributed to the conception and methodology of the study. ST performed the coding and data analysis under YX supervision, and wrote the first draft of the manuscript. All authors contributed to manuscript revision, read, and approved the submitted version.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Boyarinov, V., Fomichenko, P., Hou, J., Ivanov, K., Aures, A., Zwermann, W., et al. (2016). *Deterministic time-dependent neutron transport benchmark without spatial homogenization (c5g7-td)*. Paris, France: Nuclear Energy Agency Organisation for Economic Co-operation and Development NEA-OECD.

Boyd, W., Siegel, A., He, S., Forget, B., and Smith, K. (2016). Parallel performance results for the openmoc neutron transport code on multicore platforms. *Int. J. High Perform. Comput. Appl.* 30, 360–375. doi:10.1177/1094342016630388

Choi, N., Kang, J., and Joo, H. G. (2018). "Preliminary performance assessment of gpu acceleration module in ntracer," in *Transactions of the Korean nuclear society autumn meeting (yeosu, korea)*.

Choi, S., Kim, W., Choe, J., Lee, W., Kim, H., Ebiwonjumi, B., et al. (2021). Development of high-fidelity neutron transport code stream. *Comput. Phys. Commun.* 264, 107915. doi:10.1016/j.cpc.2021.107915

Hao, C., Xu, Y., and Downar, J. T. (2018). Multi-level coarse mesh finite difference acceleration with local two-node nodal expansion method. *Ann. Nucl. Energy* 116, 105–113. doi:10.1016/j.anucene.2018.02.002

Hou, J. J., Ivanov, K. N., Boyarinov, V. F., and Fomichenko, P. A. (2017). Oecd/nea benchmark for time-dependent neutron transport calculations without spatial homogenization. *Nucl. Eng. Des.* 317, 177–189. doi:10.1016/j.nucengdes.2017.02.008

Joo, H. G., Cho, J. Y., Kim, K. S., Lee, C. C., and Zee, S. Q. (2004). "Methods and performance of a three-dimensional whole-core transport code decart," in *Physor 2004* (Chicago, IL: American Nuclear Society).

Larsen, E., Collins, B., Kochunas, B., and Stimpson, S. (2019). "MPACT theory manual (version 4.1)," in *Tech. Rep. CASL-U-2019-1874-001*. Oak Ridge, TN, United States: (University of Michigan, Oak Ridge National Laboratory).

Quinn, M. (2003). *Parallel programming in C with MPI and OpenMP*. Boston: McGraw-Hill.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. 2nd Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Smith, K. S. (1983). "Nodal method storage reduction by nonlinear iteration"in ANS annual meeting, Detroit, MI, United States, June 12–June 17, 1983, (Transactions of the American Nuclear Society), 44, 265–266.

Tao, S., and Xu, Y. (2020). "Hybrid parallel computing for solving 3d multi-group neutron diffusion equation via multi-level cmfd acceleration," in *Transactions of the American nuclear society (virtual conference)*.

Tao, S., and Xu, Y. (2022a). Hybrid parallel strategies for the neutron transport code pandas-moc. *Front. Nucl. Eng.* 1. doi:10.3389/fnuen.2022.1002951

Tao, S., and Xu, Y. (2022b). Neutron transport analysis of c5g7-td benchmark with pandas-moc. *Ann. Nucl. Energy* 169, 108966. doi:10.1016/j.anucene.2022.108966

Tao, S., and Xu, Y. (2022c). Parallel schedules for moc sweep in the neutron transport code pandas-moc. *Front. Nucl. Eng.* 1. doi:10.3389/fnuen.2022.1002862

Xu, Y., and Downar, T. (2012). "Convergence analysis of a cmfd method based on generalized equivalence theory," in *Physor 2012: Conference on advances in reactor physics - linking research, industry, and education* Knoxville: American Nuclear Society. Apr 15-20 (TN (United States)).

Zhu, K., Kong, B., Hou, J., Zhang, H., Guo, J., and Li, F. (2022). Archer - a new three-dimensional method of characteristics neutron transport code for pebble-bed htr with coarse mesh finite difference acceleration. *Ann. Nucl. Energy* 177, 109303. doi:10.1016/j.anucene.2022.109303