An acceleration technique for 2D method of characteristics based on Krylov subspace method and CUDA technique

ZHENG Yong[1,2], and PENG Minjun[1,2*]

1. *College of Nuclear Science and Technology, Harbin Engineering University, Harbin 150001, China (heupmj@163.com)*
2. *Fundamental Science on Nuclear Safety and Simulation Technology Laboratory, Harbin Engineering University, Harbin 150001, China*

Abstract: The method of characteristics (MOC) with matrix form has more favorable performance compared with its standard application. And the linear algebraic solver affects heavily the computation efficiency. As a Krylov subspace technique, the preconditioned GMRES(PGMRES) algorithm was proposed to solve the resulting linear system in previous work. To accelerate the iteration process further, the current study utilizes the GPU-based CUDA technique to program the massively parallel PGMRES. The sparse matrix vector multiplication(SpMV), the most time-consuming operation, was optimized using the coherent visiting and shared memory, which improves the parallel computing performance significantly. Based on the analysis of the numerical feature of the coefficient matrix, two parallel optimization strategies are proposed to deal with the first part of the matrix equation, and the different schemes are utilized to the different parts of the matrix based on the size of row vector, which are expected to improve the throughput of the SpMV operation. Two benchmark problems(*i.e.* 2D C5G7 benchmark problem and 2D HTTR benchmark problem) have been simulated to verify the parallel code and measure the acceleration performance, and the numerical results demonstrate that the parallel strategies are efficient with or without CMFD method for the C5G7 problem, and the speedup can achieve more than 5.5 times with the optimal strategy for both the C5G7 and HTTR benchmark.

Keyword: method of characteristics; neutron transport; preconditioned GMRES; GPU acceleration

1 Introduction

The method of characteristics[1] (MOC) has become the most popular deterministic method to solve the neutron transport equation due to its excellent geometrical flexibility and natural parallelism. In the current reactor physics field, both the commercial and research-oriented softwares have introduced the MOC computation module to handle the arbitrary geometry for 2-D or 3-D neutron transport problems. For MOC, the computational accuracy can only be guaranteed using finer flat source regions(FSR) and dense characteristic tracks, which challenges the computational efficiency severely. The traditional source iteration of transport calculation by MOC needs to sweep these tracks' information recurrently in the scattering iterations, and these time-consuming sweeping operations result in an unacceptable slow convergence efficiency[2]. To reduce the number of source iterations, some nonlinear diffusion acceleration techniques, such as coarse mesh finite difference(CMFD) and coarse mesh rebalance method(CMR), have been utilized in many researches[3-7], and the numerical results have demonstrated that these coarse mesh methods are computationally inexpensive and efficient.

Transformed from its traditional tracks sweeping, the method of characteristics with matrix form[8-11](MMOC) provides additional possibility to accelerate the transport calculation. Contrasting with the recurrent sweepings in the traditional application, the MMOC sweeps these tracks' information only once to construct the coefficient matrix, then the remaining works are left with the linear algebraic solver. These nonzero elements in the coefficient matrix represent the response coefficients between the boundary angular fluxes and scalar fluxes in the FSRs. The matrix is only determined by the tracks' information and cross section library, which means the matrix will remain unchanged in the process of source iterations. In MMOC, the unknowns involve the scalar fluxes of FSRs and boundary incident angular fluxes. In general, the matrix pattern is unsymmetric and sparse, therefore the generalized minimal residual[12](GMRES) method is

Received date: April 29 , 2017
(Revised date: June 20, 2017)

implemented to solve the resulting sparse linear system. Actually, the GMRES has been used widely to solve the neutron transport problems due to its excellent performance[8, 13].

The restarted GMRES has the advantages of fast convergence and good stability, and there are many sustained efforts to further improve its iteration efficiency on the algorithm level[14, 15]. On the code level, the development of parallel technique provides the researchers with a new perspective to accelerate their programs further, in particular, the programmable graphics processor unit(GPU) brings the revolution of parallel computation. Traditional high performance computing depends on the large computer system or compute cluster, and all instructions are executed in the distributed central processing unit (CPU). These computation platforms are still too expensive for most researchers. Contrasting with CPU, the cost of unit computation of programmable GPU is decreasing sustainedly while its theoretical float-point operations per second(FLOPS) is several magnitudes higher than multicore CPUs[16]. In November 2006, NVIDIA introduced compute unified device architecture (CUDA), which mitigates dramatically the difficulty to program on GPU.

There are some relevant researches using the CUDA technique to accelerate GMRES[17, 18]. The kernel and most time-consuming step is the sparse matrix-vector multiplication(SpMV), which has been analysed in the reference [19] in detail. In their researches, the sparse matrix has uniform nonzeros distribution in each row in general. However, in order to balance the load for MMOC case, the additional efforts should been made to treat the irregular shape of coefficient matrix as analyzed later.

In this paper, we analysed the computation complexity of started GMRES based on the feature of coefficient matrix, and redesigned the CUDA-based GMRES with the Jacobi preconditioner. For SpMV, the original version is optimized with the coherence visiting and shared memory. Starting from this point, other two optimization strategies are proposed in the current study. The two proposed methods are verified and evaluated using two benchmark problems with a hexagonal complicated geometry, *i.e.* HTTR benchmark problem[20] and rectangular geometry, *i.e.* OECD/NEA C5G7 MOX benchmark problem[21].

The remainder of this paper is organized as follows: Section 2 exhibits the methodologies of this study in detail, including other implemented techniques unmentioned in introduction; section 3 displays the numerical experiments for the verification and evaluation of the proposed acceleration schemes; section 4 makes the conclusion for the present work.

2 Theoretical model

The current study was carried out with the following framework: the complete procedure was divided into several separate modules according to their functions. The pre-processing module was designed to handle the input file with a .xml suffix, which was written in the extensible markup language. The input file involves the configurations of computation geometry and other parameters specifying the calculation conditions as well as the acceleration techniques used in the following calculations. After the input file passed the correctness check by this pre-processing module, the geometry processing and subsequent matrix construction were performed. The geometry was represented by constructive solid geometry(CSG) formula, by which arbitrary complicated geometry can be represented. The details about geometric representation and following characteristic line tracing will not be covered in present paper, and they can be found in the reference[15].

2.1 Review of matrix characteristics method

After dividing the whole spatial domain into enormous finer FSRs, the characteristic form of transport equation can be built for each discrete neutron flight trajectory:

$$\frac{d}{ds}\psi(s)+\Sigma_t\psi(s)=q(s) \tag{1}$$

where the mark for energy group is eliminated for simplicity hereafter; s is the local coordinate along the track on X-Y plane; ψ is the neutron angular flux; Σ_t is the macro total cross section of material; and q is the neutron source including the scattering source and fission source.

With the assumption that both q and Σ_t are constants on the considered FSR i, the Eq. 1 can be solved analytically:

$$\psi_i^{out} = \psi_i^{in}\exp\left(-\Sigma_{t,i}s_i\right) + \frac{q_i}{\Sigma_{t,i}}\left[1-\exp\left(-\Sigma_{t,i}s_i\right)\right] \quad (2)$$

The Eq. 2 indicates that the outgoing angular flux ψ_i^{out} can be expressed in terms of the incident angular flux ψ_i^{in} and neutron source q_i of the FSR i. Recursively, the outgoing flux of track k on any FSRs can be obtained by its outer boundary incident angular flux ψ_k^{in} and the neutron sources q_j of all FSRs traversed by the track k:

$$\psi_{I,k}^{out} = \psi_k^{in}e_{0,I} + \sum_{j=0}^{I}\frac{q_j}{\Sigma_{t,j}}\left[1-\exp\left(-\Sigma_{t,j}s_j\right)\right]e_{j,I} \quad (3)$$

where 0 and I denote the ID numbers of the first FSR and the last FSR traversed by track k, respectively; $e_{i,j}$ is the exponential attenuation coefficient from region i to j along the track k:

$$e_{i,j} = \exp\left(-\Sigma_{t,i}s_i - \Sigma_{t,i+1}s_{i+1}\cdots-\Sigma_{t,j-1}s_{j-1}\right) \quad (4)$$

With the incident and outgoing angular fluxes ψ_i^{in}/ψ_i^{out}, the track's average angular flux $\bar{\psi}_{i,k}$ can be calculated. Furthermore, the FSR's azimuthal average angular flux $\bar{\psi}_{i,m}$ can be expressed in terms of the tracks' average angular flux and the "width" of segments. Hence the neutron scalar flux ϕ_i of FSR i can be obtained by taking weighted sum of the azimuthal average angular flux:

$$\phi_i = f_1(q_i) + \sum_k\left(f_2(\psi_k) + \sum_j f_3(q_j)\right) \quad (5)$$

where f_1, f_2 and f_3 are the response functions; q_i is the total neutron source of FSR i, involving the scattering source and fission source.

The first matrix equation of desired linear system can be obtained by splitting the in-group scattering source term from q_i and rearranging Eq. 5:

$$\left(\boldsymbol{S}_1+\boldsymbol{D}_1\right)\Phi + \boldsymbol{S}_1'\Psi = \boldsymbol{S}_1''\boldsymbol{Q} \quad (6)$$

where Φ and Ψ are the unknown vectors of scalar fluxes and boundary angular fluxes, respectively; $\boldsymbol{Q}$ is the vector consisted of out-group source term (including the scattering source from other energy groups and the fission source); $\boldsymbol{S}_1$, $\boldsymbol{S}_1'$ are the coefficient matrices with dimension of $n\times n$ and $n\times l$, respectively; $\boldsymbol{S}_1''$ can be computed from sub-matrix $\boldsymbol{S}_1$ based on the numerical properties[11]; $\boldsymbol{D}_1$ is a diagonal sub-matrix determined by cross section library; n and l are the number of FSRs and tracks, respectively.

The additional m equations to close the linear system are obtained by applying the outer boundary condition to Eq. 2, which will result in:

$$\boldsymbol{S}_2\Phi + \left(\boldsymbol{E}_2+\boldsymbol{S}_2'\right)\Psi = \boldsymbol{S}_2''\boldsymbol{Q} \quad (7)$$

where $\boldsymbol{S}_2$, $\boldsymbol{S}_2'$ are the coefficient matrices with dimension of $l\times n$ and $l\times l$, respectively; similarly, $\boldsymbol{S}_2''$ can be obtained from sub-matrix $\boldsymbol{S}_2$; $\boldsymbol{E}_2$ is an unitary matrix which implies the mapping relationship between tracks on the outer boundaries.

The Eq. 6 and Eq. 7 form the resulting algebraic system $\boldsymbol{A}x = b$, where $\boldsymbol{A}$ is stored with compress sparse row (CSR) storage format due to the considerable sparsity, and all nonzero elements are stored with single precision float point format to save the storage space and accelerate computation.

2.2 Analysis of GMRES

It is impractical to solve the resulting linear system using a direct method because of the enormous computation burden and unacceptable memory requirement. Meanwhile, the unsymmetric nonzeros distribution of the sparse matrix limits the choice of iterative method to be used. As a variant of Krylov subspace techniques, the restarted GMRES algorithm has excellent performance to solve the linear system in neutron transport field. In particular, its preconditioning version has more favorable acceleration performance than its standard application if the appropriate preconditioner is utilized[15]. The following procedure describes the standard implementation of the restarted GMRES(m) with left preconditioning which is used in the current study, where m is the Krylov subspace dimension:

1. Compute $r_0 = \boldsymbol{M}^{-1}(b-\boldsymbol{A}x_0)$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$;
2. For $j = 1, 2, \ldots, m$, Do:
3. Compute $w_j = \boldsymbol{M}^{-1}\boldsymbol{A}v_j$;
4. For $i = 1, 2, \ldots, j$, Do:
5 $h_{i,j} = \left(w_j,\ v_i\right)$;
6. $w_j = w_j - h_{i,j}v_i$;
7. EndDo
8. $h_{j+1,j} = \left\|w_j\right\|_2$, if $h_{j+1,j} = 0$ set $m = j$ and go to 11;
9. $v_{j+1} = w_j/h_{j+1,j}$;
10. EndDo
11. Define the Heisenberg matrix $\bar{\boldsymbol{H}}_m = \left\{h_{i,j}\right\}_{1\le i\le m+1,1\le j\le m}$;
12. Compute y_m the minimizer of $\|\beta e_1 - \bar{\boldsymbol{H}}_m y\|_2$;
13. Compute the latest solution $x_m = x_0 + \boldsymbol{V}_m y_m$;
14. If satisfied then Stop, else set $x_0 = x_m$, and go to 1.

In general, the subspace dimension would be smaller than 50 for the resulting sparse linear system arisen from the MMOC case, consequently, the most time-consuming step is the Arnold loop to construct the orthogonal basis while the time to yield the least square solution can be negligible. Moreover, the frequency to solve the least square problem is much inferior to the dimension m. Based on these knowledge, the optimization should be focused on the Arnold loop. Actually, our previous researches have revealed that the entire iteration efficiency depends on the cost of orthogonalization. The analysis of floating-point operations is given as the following.

It is observed that there are $2(m+1)$ SpMVs, $m(m+3)/2+1$ vector dot products, $m(m+5)/2+1$ vector scaling operations and $m(m+3)/2+1$ vector addition operations in each restart process. Supposing the length of unknown vector is N, and the number of nonzero elements in matrix $\boldsymbol{A}$ and $\boldsymbol{M}^{-1}$ are denoted by nza and nzm, respectively, the expenses to compute the approximate x_m by the preconditioning GMRES can be evaluated: taking the sum of above operations of vector-vector and vector-scalar, there are $(2m^2+7m+4)n$ floating-point operations while the number for SpMVs is $4(m+1)(nza+nzm)$. Define the sparsity as $sp=(nza+nzm)/n$ arguably and a factor f:

$$f=\frac{2m^2+7m+4}{4m+4}=1+\frac{2m^2+3m}{4m+4} \qquad (8)$$

where m belongs to the range $(0,\ 50)$ usually as mentioned before, hence the value of f can be determined in the range $(1,\ 26)$. It indicates that the vector operations will require the major time when the sparsity sp is less than f, otherwise the SpMVs will dominate the execution time exactly as that happens ordinarily. Because both the number of tracks overlaying a FSR and the number of FSRs traversed by a track are much larger than 26 for common transport problems, the average number of nonzeros per row in the coefficient matrix $\boldsymbol{A}$ is always more than 26, hence the ratio sp is mostly more than f with the subspace dimension less than 50.

It should be noted that the above theoretical analysis only consider the number of floating-point operations but neglect the accessing of operands. However, when the GMRES algorithm is implemented on machine, the practical hotspot is typically the SpMVs instead of the vector operations with regular data accessing pattern. It is because that the SpMVs suffer from the poor efficiency of fetching data rather than operating on the operands.

2.3 Optimization of GMRES based on CUDA

In recent years, the programmable GPU has evolved into a highly parallel, multithreaded and many-core processor with tremendous computational horsepower and very high memory bandwidth. In comparison with CPU, it devotes more transistors to data processing rather than the data caching and flow control as illustrated in Fig. 1. The difference in design of CPU and GPU gives them respective advantages and weaknesses, therfore the high-performance applications always take advantage of the heterogeneous parallel computing using both CPUs and GPUs: executing the sequential and complex logical parts on the CPUs and numerically intensive parts on the GPUs.

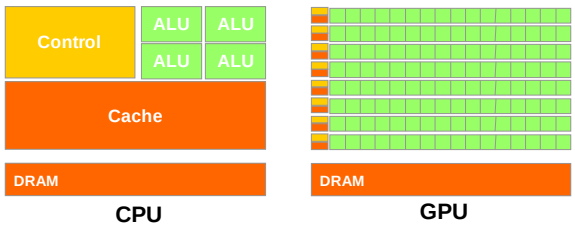


Fig. 1. Usage of transistors of CPU and GPU.

In the current study, a GPU-based GMRES algorithm written in NVIDIA's CUDA programming language is developed to solve the resulting linear system from the matrix MOC. The mentioned vector operations and SpMV in subsection 2.2 are programmed as the kernel functions which run on the device side and the remaining parts are processed on the host side. As analyzed above, the performance bottlenecks encountered by vector operations are much smaller than the SpMVs, therefore the current paper only considers the optimization to the SpMV.

The SpMV operation can be divided into two steps(namely two sub-kernels) as illustrated in Algorithm 1: firstly the nonzero elements in matrix $\boldsymbol{A}$ are multiplied by the corresponding elements in the vector and the results are kept in a temporary array $temp$; then the additive operations for each row vector are performed with $temp$. It is known

that the SpMV is memory bound as well as latency bound due to the irregular memory accessing to the vector's elements. In the first sub-kernel, the memory accessing latency can be hidden efficiently by massively activated parallel threads. For the second sub-kernel, the coherence visiting and shared memory techniques are adopted to reduce the latency caused by global memory accessing[18]. The temporal array $temp$ is divided into a number of pieces, and all operands in the current piece are loaded into a shared memory at once. It is known that the shared memory is only visible for the threads residing in the current block. After data loading, all threads residing in this block perform the additive operations of operands simultaneously. This strategy (named as gpu_cv/sm hereafter) reduces the number of memory visiting using a large-capacity shared memory. However, both the theoretical analysis and numerical experiments indicate that this optimization is only efficient for the special situation that there are only fewer nonzeros residing in each row of sparse matrix. Unfortunately, the sparsity pattern of matrix $\boldsymbol{A}$ is irregular generally as illustrated in Fig. 2 for MMOC, and there are thousands of nonzeros in some rows while only dozens of nonzeros in other rows. This situation indicates that the additive operations on temporal matrix $temp$ should be divided into two parts according to the number of nonzeros per row. Actually, this discrepancy of nonzeros arises from Eq. 6 and Eq. 7, and the sub-matrix $\boldsymbol{S}_1$ and $\boldsymbol{S}_1'$ have much more nonzeros than that in $\boldsymbol{S}_2$ and $\boldsymbol{S}_2'$.

Based on above analysis, it is obvious that different strategies should be utilized to deal with the different parts of temporal matrix $temp$ in the second step for high-performance due to the difference in nonzeros for each row. Typically, the first part arisen from Eq. 6 has more nonzeros per row as shown in Fig. 2, hence the optimization focus is on this part, and the second part will adopt the coherence visiting and shared memory technique. There are two strategies proposed to increase the throughput rate in the current paper. The first one (named as gpu_thread hereafter) is that the additive operations for each row are assigned into only one thread simply, and this thread accumulates sequentially the nonzeros belonging to a row of the temporal matrix $temp$. Since reduction of the synchronizations and branches compared with gpu_cv/sm, this method is expected to achieve a higher FLOPS. Nevertheless, it would decrease the hardware resource utilization ratio, since there are only fewer activated threads, usually equivalent to the number of rows of the coefficient sub-matrix $\boldsymbol{S}_1$, *i.e.* the number of FSRs, while the CUDA can activate massively available threads simultaneously. To overcome this drawback, the second strategy (named as gpu_block hereafter) is proposed to assign the additive operations of each row into a block, which consists of a large number of threads (up to 1024), and the number of threads per block can be specified by user and be a integer multiple of the warp size (32 for the current hardware platform) typically. The parallel reductions are implemented for the row vector of matrix $temp$ in each block, and these operands are loaded into the shared memory with low latency in advance. Comparing with the gpu_thread, this method can achieve the maximum utilization ratio of hardware resource, although there is the unbalance problem due to the inevitable variation of row sizes of the sparse matrix.

Algorithm 1 SpMV with CSR

```
1: for i = 0:nz do                                   ▷ (Step 1)
2:     temp(i) = A(i) * x(col(i));
3: end for
4: for i = 0:N do                                    ▷ (Step 2)
5:     y(i) = 0.0;
6:     start = rpos(i), end = rpos(i + 1);
7:     for j = start:end − 1 do
8:         y(i) = y(i) + temp(j);
9:     end for
10: end for
```

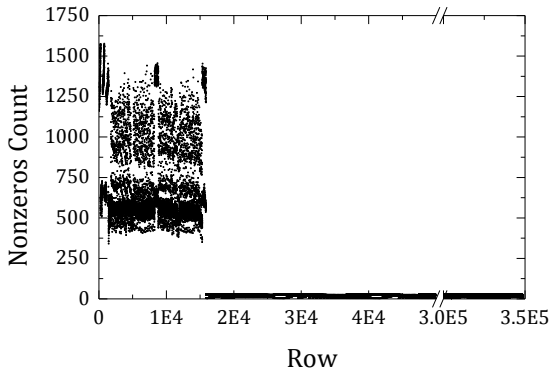


Fig. 2. Distribution for number of nonzeros per row.

3 Numerical experiments and discussions

In this section, the widely used C5G7 MOX benchmark problem[21] and the 2D HTTR problem[20] are simulated to evaluate the effectiveness of developed parallel program and measure the acceleration performance of proposed methods. The experiment environment parameters are listed in Table 1. As can be seen, all calculations are conducted on a workstation with Xeon E5-1620 3.70 GHz CPU and 32 GB RAM. This GPU comprises 13 streaming multiprocessors (SMs, a total of 2 496 CUDA cores), works at a GPU clock rate of 705 MHz and a memory clock rate of 2,600 MHz and has 5.0 GB device memory associated with a unified L2 cache of size 1.25 MB. When compiling the CUDA-based programs, the GPU architecture is specified as -arch sm_35.

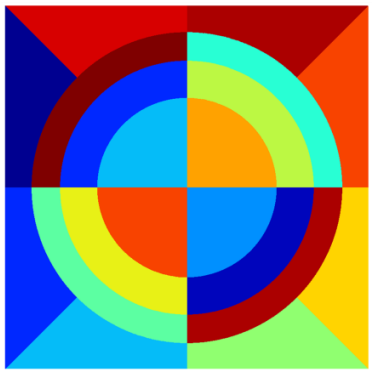
Table 1. Experiment environment parameters

Parameter	Value
CPU	Intel Xeon E5-1620
GPU	NVIDIA Tesla K20c
Operation System	Windows 7 64bit
CUDA Toolkit	6.5
Host compiler	Visual C++6.0
CPU codes compilation	-O3 option
Float point number	Single precision

3.1 The 2D C5G7 MOX benchmark problem

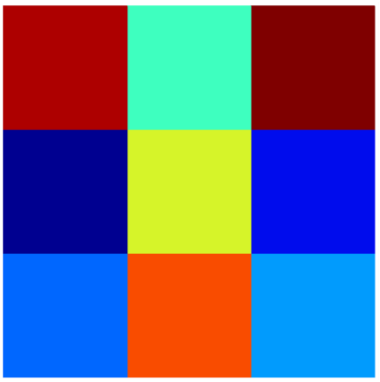
The OECD/NEA 2D C5G7 benchmark problem has been widely used to verify the deterministic transport code. This article will not cover the details of the core configuration and pin cell composition, which can be found in the reference[21]. Figure 3 gives three flat source regional discretization schemes for the pin cells in a fuel assembly and the reflector zone, respectively.

As can be seen, each pin cell in a fuel assembly consists of 8 moderator finer regions and 3 or 12 fuel-clad mixture finer regions radially with equivalent volume. To describe the dramatic gradient of scalar flux accurately, the finer division scheme in Fig. 3a is utilized to the outermost fuel cells adjacent to the reflector and the remainder of fuel cells implement the coarser discretization scheme as shown in Fig. 3b. Additionally, the two innermost reflector cells are divided into 3×3 square regions (*i.e.* 0.42 cm $\times$ 0.42 cm mesh) as illustrated in Fig. 3c and the remaining reflector cells with dimension of 1.26 cm $\times$ 1.26 cm are not subdivided any more.



(a) Finer fuel pin division

(b) Coarser fuel pin division



(c) finer reflector cell division

Fig. 3. Regional divisions of pin cell.

According to the above meshing, the computation domain comprises 15884 flat source regions generated by the geometry processing module. With respect to the discretization direction, there are two optimal Leonard polar angles[22] in the range $(0 \sim \pi/2)$ and 8 azimuthal angles in the range $(0 \sim \pi)$. The spacing of tracks is set as 0.05 cm, in addition, the convergence criteria of k_{eff} and fission power are specified as 10^{-5} and 10^{-6}, respectively.

Table 2. Accuracy assessment for C5G7 problem

Parameters	value
keff error/pcm	-83(±8)*
Average pin power error in each assembly/%	
Inner UO2	0.083(±0.101)
MOX	0.080(±0.181)
Outer UO2	-0.534(±0.195)
Specific pin power error/%	
Maximum power pin	0.078(±0.163)
Minimum power pin	-0.722(±0.582)
Maximum error	1.723(±0.250)
AVG	0.346(±0.324)
RMS	0.426(±0.337)
MRE	0.303(±0.274)
Average sigma factor	2.54

* Number in parentheses denotes the statistical error of reference solution within 98% confidence interval.

In the current study, regardless of what parallel strategy is used, the results have the same computation accuracy finally as shown in Table 2. Comparing with the results computed by other famous transport codes given in the reference[21], the present computational accuracy is acceptable. As a simple and efficient acceleration technique, the CMFD technique for rectangular lattice has been developed as a optional method in present code. The results show that the computation accuracy is independent of the CMFD technique, which just affects the number of source iterations. And there are 9 source iterations required to achieve convergence when applying the CMFD acceleration technique, otherwise 60 source iterations. The speedup performance of CMFD is remarkable for all computation conditions as illustrated in Table 3.

Table 3. Run time for the C5G7 benchmark problem[s]

	Case	CMFD	30*	25	20	15	10	5
cpu	1	W/O**	180.14	164.45	151.51	142.95	147.84	187.24
	2	With	26.71	25.83	23.34	21.12	21.70	28.84
gpu_cv/sm	3	W/O	61.24	60.40	60.01	61.30	65.89	84.64
	4	With	10.07	9.76	9.59	9.32	10.07	13.38
gpu_thread	5	W/O	34.11	33.90	33.55	33.91	36.41	45.95
	6	With	5.64	5.52	5.41	5.23	5.57	7.27
gpu_block	7	W/O	23.02	22.15	22.32	22.20	23.29	27.57
	8	With	3.69	3.59	3.54	3.38	3.57	4.47

* The number represents the Krylov subspace dimension;

** It means that the CMFD acceleration is turned off.

Table 3 gives the comparison of computation time in second when using different subspace dimensions and parallel strategies. In order to minimize the uncertainty arisen from operation system, the calculations are performed three times for each computational condition, and the results given in Table 3 are the average value. Because of the reduction of source iterations with the CMFD scheme, the computation time is reduced by more than six times for both the CPU case and the three GPU cases.

It is obvious that the GMRES(m) algorithm will need more restarting loops if a smaller subspace dimension is used, because the approximate x_m found in the smaller subspace is more far away from the exact solution than that found in a larger subspace. Moreover, the new subspace is hardly orthogonal with the previous one exactly, so the restarted GMRES(m) algorithm with a smaller subspace dimension will need not only more restarting loops but also more iterations (SpMVs). On the other hand, the larger subspace dimension means that more vector operations are performed, and the quadratic scaling feature doesn't allow one to choose too large subspace dimension. Based on this analysis, there is an optimal subspace dimension for each problem, as shown in Table 3. The influence of subspace dimension on computation efficiency is more significant in the CPU versions than that in the GPU versions, since the time of vectors operations in GPU versions is ignorable by taking advantage of massively parallel and the regular accessing to the memory. Despite this, there is an optimal subspace dimension 15 arguably for the C5G7 benchmark problem.

Figure 4 displays the comparison of speedups for different computation conditions. There is a

considerable difference in speedup among the implemented parallel strategies. The CMFD method has little influences on the speedup for each parallel strategy, the speedups are doubled with the improvement of parallel strategy. There are some variations in speedup with the increase of subspace dimension, it is mainly caused by the CPU cases time which is more sensitive to the subspace dimension than GPU cases.

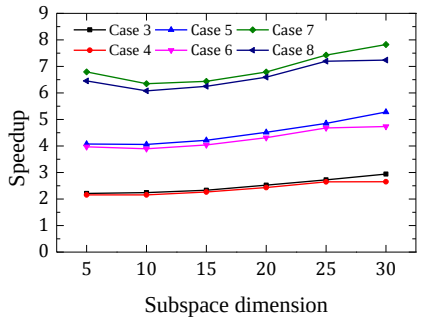
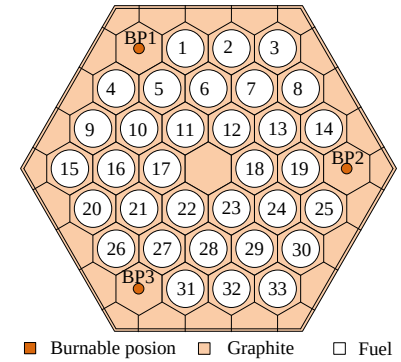


Fig. 4. Speedup for difference computation condition.

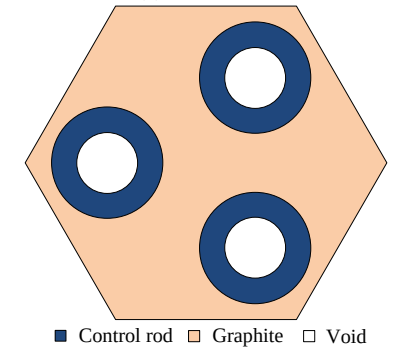
3.2 The 2D HTTR benchmark problem

This problem is extracted from the reference[20], in which both the 2D and 3D benchmark problems are developed to examine the capacity of deterministic transport code. Its prototype is the experimental High Temperature Engineering Test Reactor (HTTR), which was built by JAERI (Japan Atomic Energy Research Institute) in the late 1990s. The lattice code HELIOS was used to generate the 6-group macroscopic cross section library, and the reference solutions were calculated by MCNP with the generated cross section library.

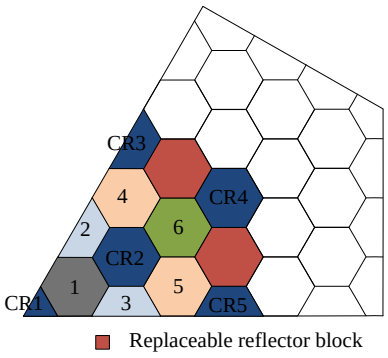
Due to the symmetry design, only one sixth of the core is modeled in present study. There are three geometry configurations according to the insertion patterns of the control rods(case 1: all-rods-in; case 2: partially-controlled; case 3: all-rods-out). Figure 5 gives the geometry configurations of fuel block, control block and the 1/6 core, all necessary geometric dimensions are listed in Table 4. There are four fuel types/enrichments marked with different colors as illustrated in Fig. 5c, and each fuel block consists of 33 fuel cells and 3 burnable poison rods as shown in Fig. 5a. The control rod is annular, and there is a coolant hole filled with helium at the center of control rod. Since the helium is not included in the cross section library of the code HELIOS, the channels filling with helium are replaced by void in the current and reference calculations. The partially-controlled case means that all control rods in CR3, 4 and 5 are inserted while CR1 and CR2 are withdrawn as shown in Fig. 5c.



(a) fuel block



(b) control block



(c) block indexing in 1/6 core

Fig. 5. Block structure and 1/6 core configuration of HTTR.

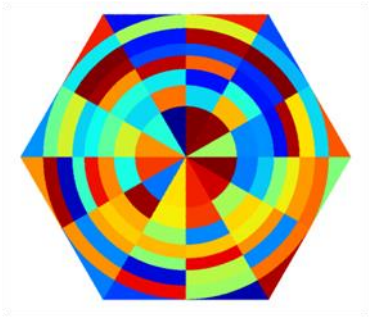
Table 4. Simplified HTTR benchmark geometry parameters

Flat-to-flat core width	436.4768 cm
Flat-to-flat block width	36 cm
Fuel pin (BP rod) pitch	5.15 cm
Fuel pin diameter	4.1 cm
BP rod diameter	1.5 cm
Control rod diameter	12.3 cm
Control rod inner diameter	6.7 cm
Distance from control rod center to block center	10.8 cm

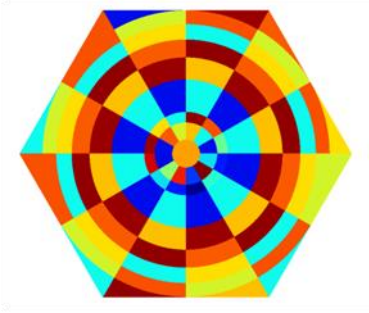
Figure 6 displays the flat source regions discretization schemes for the fuel cell, burnable poison cell and reflector cell. Since the variation of neutron flux gradient, the discretizations of reflector cells are different for the replaceable reflector block and permanent reflector block. There are 18 reflector cells as shown in Fig. 6c in a replaceable reflector block between the opposite sides while 16 and 10 reflector cells in the inner and outer permanent reflector block, respectively. Overall, there are 50 110, 49 670 and 48 916 flat source regions generated in the one sixth core for three control rods configurations, respectively. The other discrete parameters are specified as the previous subsection.

The computation results for the 2D HTTR benchmark problem are listed in Table 5, and it should be pointed out that the time does not include the consumptions of geometry processing and coefficient matrix construction, and only the time of source iterations are given here. The results show that the parallel strategies and the subspace dimension have little or no influence on the computation accuracy and the number of source iterations for all three cases, and the trivial variations of source iterations in the gpu_block are mainly caused by the loss of significance when performing the parallel reduction operations. The linear system solution time depends upon the Krylov subspace dimension heavily, especially for the CPU version. For both the gpu_cv/sm and gpu_thread cases, the dimension no longer has any pronounced influence on the runtime when the dimension is greater than 30. This is because there are more iterations(proportional to the number of SpMVs) required in the GMRES algorithm with a smaller dimension. Once the dimension is greater than 30, the number of GMRES iterations will hardly change any more and the time of vector operations in the GMRES algorithm can be neglected as mentioned before.

Figure 7 displays the comparison of speedups under different computation conditions for the three insertion patterns of the HTTR benchmark problem. There is a tendency that the speedup increases with the subspace dimension, and this phenomenon results from the sensitivity of solution time to the dimension in CPU version while the solution time in GPU case is practically independent of the dimension greater than 30 as mentioned above. Since introducing the uncertainty of the rounding errors when performing the SpMVs, the speedup tendency of the gpu_block is not regular like the previous two cases. But overall, there is a significant improvement in speedup compared with the previous two cases for all three control rods configurations.



(a) fuel cell



(b) burnable poison cell

(c) reflector cell

Fig. 6. Flat source regions partition schemes.

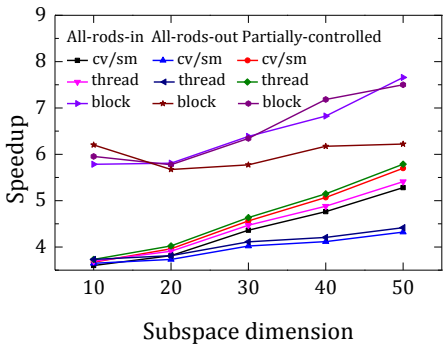


Fig. 7 Speedups comparison for HTTR benchmark problem.

Table 5. Computation results for 2D HTTR benchmark problem

M*	cpu			gpu_cv/sm			gpu_thread			gpu_block		
	Error** /pcm	Time/s	Itrs.	Error /pcm	Time/s	Itrs.	Error /pcm	Time/s	Itrs.	Error /pcm	Time/s	Itrs.
Case 1: All-rods-in												
10	-134.2	3068.02	203	-134.3	852.18	204	-134.3	830.78	204	-133.8	530.43	207
20	-134.3	2832.36	204	-134.3	743.02	204	-134.3	724.78	204	-133.8	487.61	202
30	-134.4	3058.19	205	-134.4	700.96	205	-134.4	684.27	205	-133.9	478.80	207
40	-134.4	3300.69	205	-134.3	692.93	204	-134.3	676.55	204	-134.0	483.65	211
50	-134.4	3678.78	205	-134.3	696.76	204	-134.3	680.03	204	-134.2	480.43	208
Case 2: Partially controlled												
10	7.3	3006.82	193	7.3	818.94	194	7.3	805.89	194	7.7	504.99	197
20	7.3	2744.51	194	7.3	692.80	194	7.3	682.32	194	8.0	475.22	200
30	7.3	2962.06	194	7.3	649.27	194	7.3	639.13	194	8.6	467.06	208
40	7.3	3269.36	194	7.3	645.18	194	7.3	635.23	194	7.6	455.08	201
50	7.3	3680.06	194	7.3	645.70	194	7.3	635.91	194	7.6	490.56	220
Case 3: All-rods-out												
10	-100.5	3558.87	257	-100.4	975.65	256	-100.4	953.60	256	-98.0	573.69	245
20	-100.4	3056.14	256	-100.5	818.83	257	-100.5	801.45	257	-99.0	538.80	251
30	-100.5	3011.01	257	-100.5	749.14	257	-100.5	732.55	257	-100.1	521.51	255
40	-100.5	3084.66	257	-100.5	749.33	257	-100.5	733.40	257	-97.8	499.69	244
50	-100.5	3237.48	257	-100.5	749.42	257	-100.5	733.21	257	-98.9	520.32	252

* The M represents the subspace dimension;

** The error represents the relative error of k_{eff} comparing with the reference solution.

4 Conclusion

In this paper, the preconditioned GMRES algorithm was implemented to solve the resulting sparse linear system arisen from the MMOC method. To find the hotspot, the computation complexity of the algorithm has been analyzed, and the theoretical result shows that the SpMV operations require the majority of runtime with the typical Krylov subspace dimension. The GPU-based CUDA technique was utilized to accelerate the linear system solution process using the coherent accessing and shared memory technique. However, the feature of coefficient matrix would decrease the throughput rate of parallel code, especially the SpMV operation, since there is the difference in the row size of the coefficient matrix. Based on this knowledge, another two optimal parallel strategies focusing on the first part of the matrix are proposed to perform the accumulation operations of a row vector within a thread and threads block, respectively. The C5G7 benchmark problem and the 2D HTTR hexagonal benchmark problem have been simulated to verify the parallel code and valuate the acceleration performance of the proposed methods. The results indicate that the parallel strategies have no pronounced influence on the computation accuracy, and the proposed optimal schemes have higher speedups, in particular, the second scheme has a remarkable improvement with respect to the acceleration performance since more hardware resource was utilized than the first optimal scheme.

References

[1] ASKEW, J. R.: A characteristics formulation of the neutron transport equation in complicated geometrys, United Kingdom Atomic Energy Authority Reactor Group,1972.

[2] ZHANG, Z., LI, Q., and WANG, K.: Parallelization method for three dimensional MOC calculation, Atomic Energy Science and Technology, 2013, 47(Suppl.): 38-42. (In Chinese)

[3] CHO, J. Y., KIM, K. S., SHIM, H. J., SONG, J. S., LEE, C.-C., and JOO, H. G.: Whole Core Transport Calculation Employing Hexagonal Modular Ray Tracing and CMFD Formulation, Journal of Nuclear Science and Technology, 2008, 45(8): 740-751.

[4] HAN, Y., JIANG, X., and WANG, D.: CMFD and GPU acceleration on method of characteristics for hexagonal cores, Nuclear Engineering and Design, 2014, 280: 210-222.

[5] TANG, C., and ZHANG, S.: Development and verification of an MOC code employing assembly modular ray tracing and efficient acceleration techniques, Annals of Nuclear Energy, 2009, 36(8): 1013-1020.

[6] CAI, X., YAO, D., and WANG, K., *et al*.: Generalized coarse-mesh finite difference acceleration for method of characteristics in unstructured meshes, Chinese Journal of Computational Physics, 2010, 27(4): 541-546. (In Chinese)

[7] YAMAMOTO, A.: Generalized Coarse-Mesh Rebalance Method for Acceleration of Neutron Transport Calculations, Nuclear Science and Engineering, 2005, 151: 274-282.

[8] ZHANG, H., WU, H., and CAO, L.: An acceleration technique for 2D MOC based on Krylov subspace and domain decomposition methods, Annals of Nuclear Energy, 2011, 38(12): 2742-2751.

[9] ZHANG, H., ZHENG, Y., WU, H., and CAO, L.: A 2D/1D coupling neutron transport method based on the matrix MOC and NEM methods. In: International Conference on Mathematics and Computational Methods Applied To Nuclear Science and Engineering. Sun Valley, United Stated. 2013.

[10] LIU, Z., COLLINS, B., KOCHUNAS, B., DOWNAR, T., XU, Y., and WU, H.: Theory and analysis of accuracy for the method of characteristics direction probabilities with boundary averaging, Annals of Nuclear Energy, 2015, 77: 212-222.

[11] WU, W., LI, Q., and WANG, K.: Matrix method of characteristics based on modular ray tracing, Nuclear Power Engineering, 2014, 35(3): 129-132. (In Chinese)

[12] SAAD, Y., and H.SCHULTZ, M.: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on scientific and statistical computing, 1986, 7(3): 856-869. TURCKSIN, B., RAGUSA, J. C., and MOREL, J. E.: Angular Multigrid Preconditioner for Krylov-Based Solution Techniques Applied to theSnEquations with Highly Forward-Peaked Scattering, Transport Theory and Statistical Physics, 2012, 41(1-2): 1-22.

[13] TAKEDA, S., and KITADA, T.: Development of efficient Krylov preconditioning techniques for multi-dimensional method of characteristics, Journal of Nuclear Science and Technology, 2012, 49(4): 457-465.

[14] ZHENG, Y., PENG, M., and XIA, G.: Development and verification of MOC code based on Krylov subspace and efficient preconditioning techniques, Annals of Nuclear Energy, 2017, 99: 427-433.

[15] [NVIDIA: CUDA C Programming Guide, NVIDIA Corporation,2016.

[16] RAPHAEL, C., and STEPHANE, D.: Sparse systems solving on GPUs with GMRES, J. Supercomput., 2012, 59(3): 1504-1516.

[17] LIU, Y., YIN, K., and WU, E.: Fast gmres-gpu solver for large scale sparse linear systems, Journal of Computer-Aided Design and Computer Graphics, 2011, 23(4): 553-560. (In Chinese)

[18] BELL, N., and GARLAND, M.: Efficient sparse matrix-vector multiplication on CUDA, NVIDIA Corporation,2008.

[19] ZHANG, Z., RAHNEMA, F., ZHANG, D., POUNDERS, J. M., and OUGOUAG, A. M.: Simplified two and three dimensional HTTR benchmark problems, Annals of Nuclear Energy, 2011, 38(5): 1172-1185.

[20] NEA/OECD: Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation(A 2-D/3-D MOX Fuel Assembly Benchmark), Nuclear Energy Agency/Organization for Economic Co-operation and Development,2003.

[21] LEONARD, A., and MCDANIEL, C. T.: Optimal polar angles and weights for the characteristics method, Transactions of the American Nuclear Society, 1995, 73: 172-173.