Development of GPU Based Depletion Module by CRAM

Woo Kyoung Ko and Hyung Jin Shim
Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea
**Corresponding author: shimhj@snu.ac.kr*

1. Introduction

Parallel computation on GPU has been receiving attention from the researchers on the reactor physics computations. It is suited to problems where the same calculation is applied to numerous numbers of data. As well as the Monte Carlo neutron transport in which neutrons are simulated in parallel based on the Boltzmann transport equation, the depletion calculation also fits on this property in that numerous depletion areas are processed by solving the burnup equation.

There has been research [1,2] to apply the GPU parallel computation to the depletion calculation. Heimlich et al. [2] developed a GPU depletion solver based on the two variants of Runge-Kutta methods with optimum fine step calculation. Kim et al. [1] developed a scalable GPU depletion solver based on the Chebyshev rational approximation method (CRAM) [3] with a Gauss-Seidel linear solver. Still, there is no research in which both the depletion region and the depletion calculation of each region are parallelized.

In this paper, a depletion module on GPU based on the matrix exponential by the CRAM is introduced and validated. The purpose of this research is to find what parallelism condition gives the optimum result for the solver on GPU when both the depletion regions and the equation solver per region are parallelized. The Jacobi method, in which the calculation can be easily parallelized, is used to solve linear systems that appear in the CRAM calculation.

2. Model and Methods

In this section, the methods applied to the depletion module are introduced. Then the model used for the validation of the module is described.

2.1 Matrix Exponential Method

The burnup equation is a master equation that describes the depletion of a material over time by the decay of the consisting nuclides and the reaction with the neutron, which can be written as Eq. (1).

$$\frac{d\vec{N}(t)}{dt} = \mathbf{A}\vec{N}(t), \tag{1}$$

where t is the time, $\mathbf{A}$ is a burnup matrix in which the decay constant and the reaction rates of each nuclide are included, and the vector $\vec{N}(t)$ is the density of nuclides that consist of the material of our interest at the specific region at time t.

The general solution of Eq. (1) can be written as Eq. (2), which includes a matrix exponential.

$$\vec{N}(t_0 + t) = e^{\mathbf{A}t}\vec{N}(t_0), \tag{2}$$

where t_0 denotes the initial time where we know the initial density $\vec{N}(t_0)$.

2.1 CRAM

CRAM is an approximation method for calculating the matrix exponential, based on the residue integration and the rational approximation. It is suitable for the problem in which all its eigenvalues of the matrix on exponent are on or close to the real axis of the same sign. The suitability of the CRAM to the depletion calculation has been well investigated. [3] Following the partial fraction decomposition form [3] of the *K*-th order CRAM, the equation (3) must be solved to calculate the matrix exponential.

$$e^{\mathbf{A}t}\vec{N}(t_0) \approx \alpha_0\vec{N}(t_0) + 2\,\mathrm{Re}\left(\sum_{k=1}^{K/2}\alpha_k(\mathbf{A}t - \theta_k\mathbf{I})^{-1}\vec{N}(t_0)\right), \tag{3}$$

where the term θ_k and α_k are the pre-calculated constants that corresponds to the pole and residue of the k-th rational approximation term respectively.

2.3 Jacobi Method and Its Parallel Implementation

The typical burnup matrix is sparse; its row contains few entries below the order of hundreds, and most of their number is below 40. Therefore, row-wise parallelization is beneficial to the performance of the calculation over column-wise parallelization. The Jacobi method is capable of this parallelization because its row-wise calculation has no dependence on the value of each other within a step of iteration.

For each order k, to solve the linear system $A^{(k)}\vec{x}^{(k)} = \alpha_k\vec{N}(t_0)$ for $\vec{x}^{(k)}$, the Jacobi method is applied to k systems in parallel On each step of the Jacobi iteration, it processes each row by a thread within the thread block by iterating through a virtual 'chunk' of the rows, of which its size is less than or equal to the thread block. This approach enables a simple thread-block-wise synchronization to be performed instead of multiple calls to the expensive device-wise synchronization.

After a step of calculation is concluded, the following Eqs. (4)(5) are used to determine the convergence between two successive steps $\vec{x}^{(k,i)}$ and $\vec{x}^{(k,i+1)}$.

$$|\vec{x}^{(k,i+1)} - \vec{x}^{(k,i)}| \le \varepsilon_{rel}\,|x^{(k,i)}|, \tag{4}$$

$$\vec{x}^{(k,i+1)} - \vec{x}^{(k,i)}| \le \varepsilon_{abs}, \tag{5}$$

where i is an index of the current step, ε_{rel} is a relative tolerance, and ε_{abs} is an absolute tolerance for the

convergence test. If Eq. (4) or Eq. (5) is satisfied, the solver assumes the vector $\vec{x}^{(k)}$ has converged at step *k*.

The results whether each row satisfied Eq. (4) or Eq. (5) are collected by the CUDA synchronization API '__syncthreads_and()', which returns a non-zero integer only when every thread inside the block sends the non-zero integers. The maximum iteration count is set to 1000, and the function ends if either the solution converged or the iteration count hit the limit.

2.4 PWR Pin Cell Model

A typical PWR fresh fuel pin cell is selected to validate the CRAM GPU module. The model is supplied from the example input in the user manual of the Monte Carlo neutron transport code McCARD [4]. Its geometric configuration and scales are specified in Fig. 1.

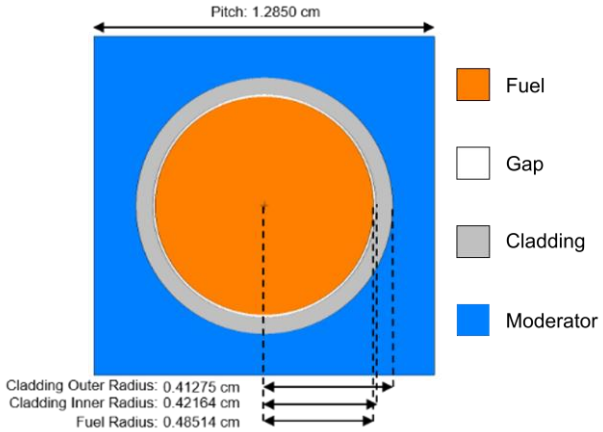

Fig. 1. Geometric specification of the PWR pin cell model.

The boundary condition of each side of the moderator is reflective.

Table I describes the initial properties and conditions of the fresh fuel pin cell.

Table I: Initial pin cell properties

Material	Fuel	UO_2
	Gap	Air
	Cladding	Zr
	Moderator	H_2O
Temperature (K)	Fuel, Gap, Cladding	870.00
	Moderator	585.22
U-235 Enrichment (wt%)		3.36
Boron Concentration (ppm)		1000
Power Density (W/g_{HW})		36.875
Power (MW)		3.54067
Burnup time step (s)		86400

Table II describes the initial configuration of the materials used in the fresh fuel pin cell input. Note that the depletion process is only applied to the UO_2.

Table II: Initial material configuration

Material	Density (g/cm^3)	Nuclide	Weight Fraction (%)
UO_2	10.176	O-16	11.8500
		U-234	0.0237
		U-235	2.9618
		U-238	85.1645
Air	0.001	O-16	100.0000
Zr	6.550	Zr-90	51.4500
		Zr-91	11.2200
		Zr-92	17.1500
		Zr-94	17.3800
		Zr-96	2.8000
H_2O	0.700	H-1	11.1900
		O-16	88.8100
		B-10	0.0020
		B-11	0.0080

The depletion matrix was constructed based on the reaction rates in the fuel pin, which were calculated by the McCARD. Based on the decay library supplied with the McCARD distribution by default, the depletion matrix includes 1695 nuclides. Figure 2 shows the distribution of non-zero entries in the constructed depletion matrix.

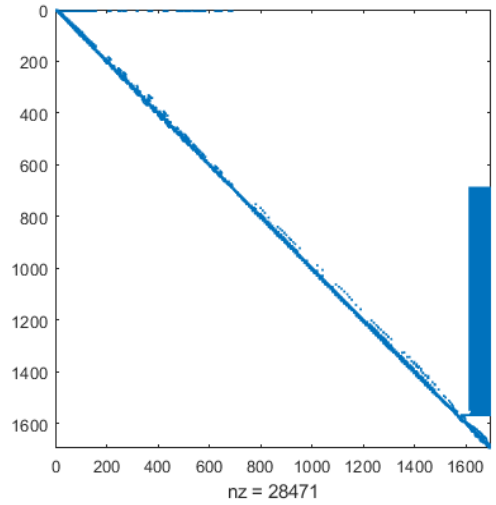

Fig. 2. Sparsity pattern of the depletion matrix generated from the PWR pin cell model

The variable 'nz' on Fig. 2 denotes the number of non-zero entries of this matrix. The matrix is extremely sparse, only having 0.99% of the non-zero terms. Therefore, the memory for this matrix can be saved by storing the matrix into a sparse format. For example, by converting the matrix into the CSR format, which compresses a matrix into three arrays, the 21.9 MB of memory required for the full double matrix can be reduced to 340 KB, saving 98.5% of the memory space.

3. Validation Results

In this section, the implemented GPU solver is validated by the reference solutions. Only the range where the atomic number density is greater than 1.0 #/cm^3 = 10^{-24} #/barn/cm are considered. This is for including Eq. (5) as the convergence criteria, which can reduce the step needed to converge the solution in the range of our interest while neglecting the nuclides of density that are physically meaningless.

4.1 Reference Solutions

As reference solutions, two solutions based on different methods were calculated to validate the GPU depletion module. One is a 16th-order CRAM solver run on a CPU, in which its linear solver uses Gaussian elimination. The solver is provided by the McCARD as an option. This result will be notated as 'CPU CRAM' in this paper. The other is a solution calculated on MATLAB® with 100 significant digits using Symbolic Math Toolbox™. The built-in function 'expm', which internally utilizes the Padé approximation with the scaling and squaring technique, was used to calculate the solution. [6] This solution is expected to be the most accurate result.

The relative error of the CPU CRAM to the MATLAB expm solution is plotted on Fig. 3.

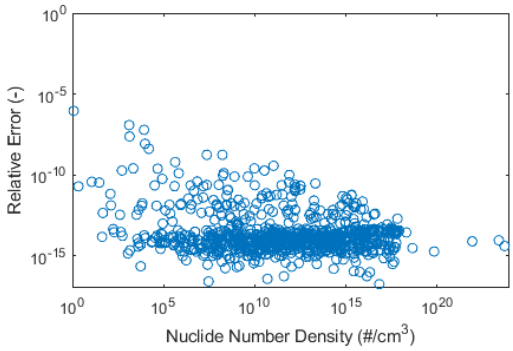

Fig. 3. Relative error of the traditional matrix exponential method and CRAM implmentation on CPU, compared to the MATLAB expm solution.

Above the density 1.0 #/cm^3, the CPU CRAM has a relative error of 9.00×10^{-7} at maximum, and most errors are below the order of 10^{-10}.

4.2 Relative Tolerance

First, the relative tolerance ε_{rel} in Eq. (4) used by the GPU Jacobi CRAM was varied and tested. Figure 4 shows the relative error of the solutions compared to the CPU CRAM using the Gaussian elimnation. The absolute tolerance ε_{abs} in Eq. (5) was not considered here, by setting the $\varepsilon_{abs} = 0$.

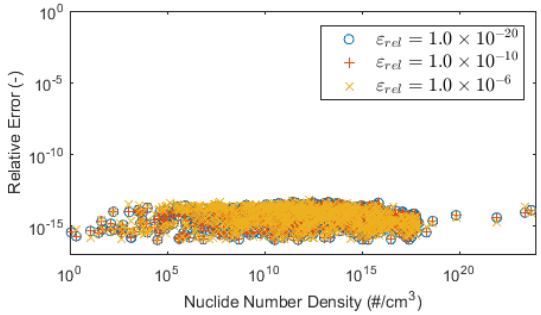

Fig. 4. Relative error of the GPU Jacobi CRAM solutions, compared to the result from the CPU CRAM, depending on the relative tolerance.

Regardless of the relative tolerance, the Jacobi solver calculated the result close to the CPU CRAM solver using the Gaussian elimination. Considering that the double-precision floating-point can only express its significant digits around 15 to 17, this indicates that two solvers calculated roughly equal solutions.

Next, the Fig. 5 compares the results from the GPU Jacobi CRAM to the MATLAB expm solution. The solution from the CPU CRAM is also plotted.

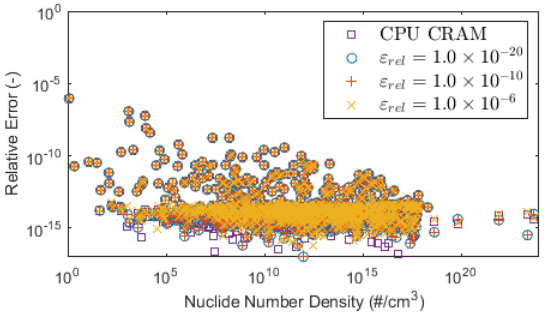

Fig. 5. Relative error of the GPU Jacobi CRAM solutions, compared to the MATLAB expm result, depending on the relative tolerance.

The overall results from GPU Jacobi CRAM shared similar behavior with the result from the CPU CRAM. Compared to the EXPM, all relative tolerance conditions including the CPU CRAM gave a similar order of the maximum error, 9.00×10^{-7}.

4.3 Absolute Tolerance

Next, the absolute tolerance ε_{abs} in Eq. (5) was varied, while the relative tolerance ε_{rel} in Eq. (4) was fixed to 10^{-20} and 10^{-6}, respectively. It is to validate that the use of the absolute tolerance will not significantly affect the accuracy of the solution inside the range of densities of our interest.

Figure 6 shows the relative error of each conditions compared to the CPU CRAM using the Gaussian elimination.

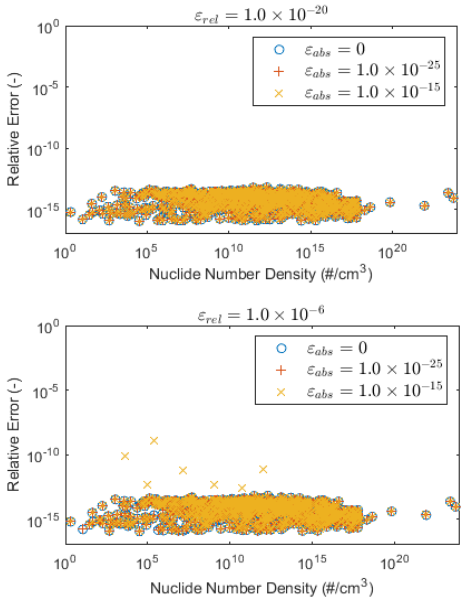

Fig. 6. Relative error of the GPU Jacobi CRAM solutions, compared to the result from the CPU CRAM, depending on the absolute tolerance.

Condition $\varepsilon_{abs} = 0$, marked with the blue circles in Fig. 6, is the same condition as the case tested in section 4.2, which is also marked with the blue circles in Fig. 4. Inside the given range, almost all solutions by the Jacobi CRAM gave the same relative error, except the condition in which $\varepsilon_{rel} = 1.0 \times 10^{-6}$ and $\varepsilon_{abs} = 1.0 \times 10^{-15}$.

Figure 7 shows the relative error of each conditions compared to the EXPM.

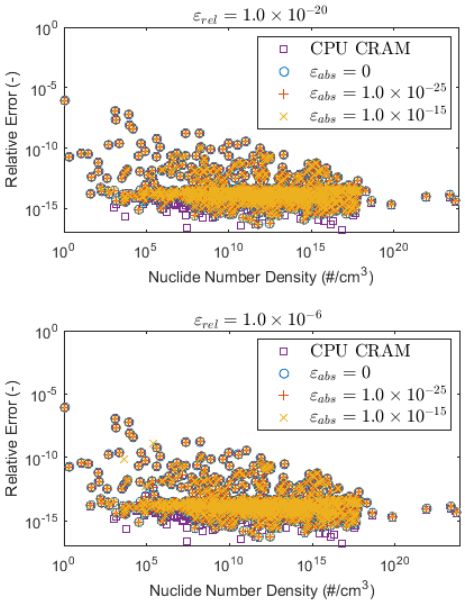

Fig. 7. Relative error of the GPU Jacobi CRAM solutions, compared to the MATLAB expm result, depending on the absolute tolerance.

Although the condition in which $\varepsilon_{rel} = 1.0 \times 10^{-6}$ and $\varepsilon_{abs} = 1.0 \times 10^{-15}$ gave the less accurate result in Fig. 6, the order of its relative error to the MATLAB expm is similar to that of other convergence conditions around the density where discrepancy happened.

4. Performance Analysis

Two main parameters were varied to determine their optimum values in the GPU CRAM depletion module: the convergence conditions in Table III and thread number per block. The thread block number determines both the speed and occupancy of the GPU. Low GPU occupancy means the device is underutilized, but it also implies that more solvers can run on the device simultaneously. The performance tests in this section were conducted on the system described in Table III.

Table III: Computing system for profiling

CPU	13th Gen Intel® Core™ i5-13500 2.50GHz
RAM	32.0 GB
GPU	NVIDIA GeForce RTX 4070
VRAM	12.0 GB
OS	Windows 10 19044.3208

4.1 Convergence Check Tolerance

Table IV is about the number of steps required for each Jacobi solver to decide whether the solution of 8 CRAM-induced systems has converged.

Table IV: Convergence step used in Jacobi CRAM

k		1	2	3	4	5	6	7	8
ε_{abs}	ε_{rel}								
0	1.0E-06	109	113	115	118	120	121	123	122
	1.0E-10	109	113	115	118	120	121	123	122
	1.0E-20	109	113	115	118	120	121	123	122
1.0E-25	1.0E-06	15	16	16	17	17	18	18	18
	1.0E-10	16	16	16	17	17	18	18	18
	1.0E-20	19	19	19	19	19	19	19	20
1.0E-15	1.0E-06	15	15	15	15	15	15	16	16
	1.0E-10	16	16	16	16	16	17	17	17
	1.0E-20	19	19	19	19	19	19	19	20

The variable *k* denotes the index of the eight matrices generated from the 16th-order CRAM. Introducing the absolute tolerance in any order decreased the step needed for the calculation. Meanwhile, the change in the relative tolerance had little effect on the convergence step count.

Figure 6 shows the average runtime of completing a single CRAM calculation for the given input, with its convergence condition changing. Each calculation was repeated ten times.

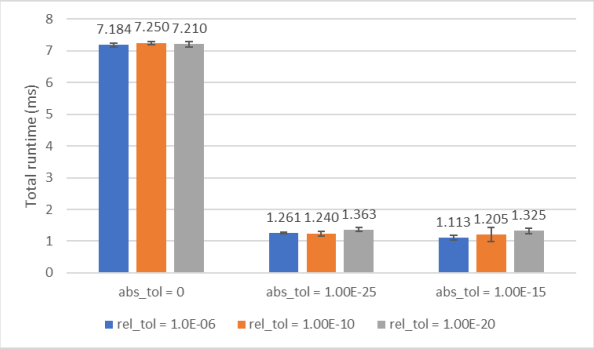

Fig. 8. Total runtime to complete a whole calculation with thread block of size 1024, with regards to the convergence conditions. Time spent for memory copy is included.

As expected from the number of steps in Table IV, the existence of the absolute tolerance had a noticeable effect on the performance, while the difference in the relative tolerance made only a slight change to the runtime.

4.2 GPU Thread Usage and Occupancy

As mentioned in section 2.3, the module utilizes *k*/2 thread blocks for the *k*-th order CRAM, and each thread block solves one linear system by the Jacobi method. Selecting an optimal number of threads per block will determine the performance and scalability of this solver. More threads per block will reduce the runtime of the parallel calculation, but each solver will consume more registers and resources, decreasing the possible occupancy achievable on the GPU. On the other hand, a smaller block size implies that more work is allocated to each thread, possibly deteriorating the performance of the solver on one individual problem. However, this will increase the available resources on the GPU, thus processing more inputs per device in parallel.

Table V summarizes the total runtime taken to finish one time step of depletion calculation, the theoretical occupancy of the GPU, and the actual occupancy achieved with regards to the thread number. The performance test was conducted based on the convergence condition $\varepsilon_{rel} = 1.0 \times 10^{-20}$ and $\varepsilon_{abs} = 1.0 \times 10^{-25}$.

Table V: Runtime and GPU occupancy

Thread number	Runtime (ms)	GPU Theoretical Occupancy (%)	GPU Achieved Occupancy (%)	Estimated Runtime per region (ms)
1	75.469	50.0	2.08	3.145
16	10.867	50.0	2.08	0.453
32	6.195	50.0	2.08	0.258
64	3.623	83.3	4.17	0.181
128	2.219	83.3	8.33	0.222
256	1.475	83.3	16.67	0.295
512	1.438	66.7	33.33	0.719
1024	1.326	66.7	66.67	1.326

The thread numbers over 256 do not have a significant difference in performance between each other, but their occupancies differ. Using 1024 threads filled the theoretical occupancy of 66%. It implies that more kernels cannot be processed simultaneously on this GPU, and the usage of the GPU is limited to 66% of its full ability. Meanwhile, with 256 threads, a similar performance can be achieved compared to the case of 1024 threads while deriving a higher theoretical occupancy and lower achieved occupancy, which makes four more kernels to be processed simultaneously. Assuming that all the possible resources are allocated to GPU kernels to run them simultaneously, therefore fully achieving the theoretical occupancy, the best result is expected from the case of 64 threads, whose effective runtime takes 0.181 ms per region. However, using fewer threads lower than 64 decreased the theoretical occupancy, achieving the same occupancy as the case of 32 threads, and their effective runtimes for each region deteriorated.

5. Conclusions

The depletion solver based on the CRAM and the Jacobi method was implemented. When setting the convergence criteria of the iterative solver in the CRAM, a strict relative tolerance is recommended in that they don't significantly affect the performance. The existence of the absolute tolerance helped to reduce the iteration step needed to converge the solution by skipping the convergence test for the density below the scale of our interest. If the available GPU resources are limited, assigning more threads per solver was beneficial to increase individual performance. A low number of threads, at least over 32, is recommended to reduce the effective runtime of each depletion calculation over the multiple depletion regions.

REFERENCES

[1] K.M. Kim, N. Choi, H.G. Lee, H.G. Joo, Practical methods for GPU-based whole-core Monte Carlo depletion calculation, Nuclear Engineering and Technology, Vol 55, p. 2516, 2023.
[2] A. Heimlich, F.C. Silva, A.S. Martinez, Fast and accurate GPU PWR depletion calculation, Annals of Nuclear Energy, Vol 117, p. 165, 2018.
[3] M. Pusa, Higher-Order Chebyshev Rational Approximation Method and Application to Burnup Equations, Nuclear Science and Engineering, Vol 182, p. 297, 2016.
[4] H.J. Shim, B.S. Han, J.S. Jung, H.J. Park, C.H. Kim, MCCARD: MONTE CARLO CODE FOR ADVANCED REACTOR DESIGN AND ANALYSIS, Nuclear Engineering and Technology. Vol 44, p. 161, 2012.
[5] H.G. Lee, H.G. Joo, Optimization of the GPU-Based Depletion Solver in nTRACER, Transactions of the Korean Nuclear Society Autumn Meeting, p 17–18, 2020.
[6] Al-Mohy, A. H. and N. J. Higham, A new scaling and squaring algorithm for the matrix exponential, SIAM Journal on Matrix Analysis and Applications, Vol 31, No. 3 p. 970, 2009.