Optimization of the GPU-Based Depletion Solver in nTRACER

Han Gyu Lee and Han Gyu Joo*
Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea
**Corresponding author: joohan@snu.ac.kr*

1. Introduction

Massive parallelization with graphics processing units (GPU) had already been applied in nTRACER, which demonstrated substantial performance [1]. It successfully accelerated the hotspots of the steady-state calculation module, especially the planar method of characteristics (MOC). Recently, a more extensive GPU offloading had been made to accelerate the entire procedure for whole-core depletion calculations [2].

However, previous research could not extract fully the performance of GPUs for depletion calculations. As the main memory of a GPU is not shared with CPUs, explicit communications of the memory caused non-negligible overheads. Furthermore, having an optimal data structure for GPU acceleration is not necessarily optimal for CPU parallelization. Due to such reasons, the depletion solver had shown less improvements than other calculations.

This leads to the large portion of depletion calculations during a burnup step as **Figure 1**. In addition, the detailed profile of depletion calculations revealed the stagnation from overheads as much as half of the total time. It was necessary to lessen those overheads, while accelerate the main solution processes.

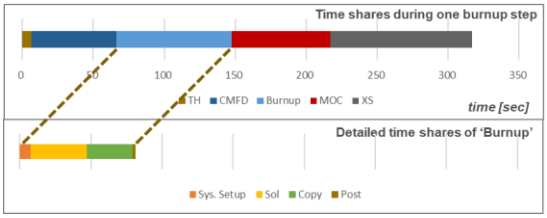

Figure 1 Time share in a depletion step.

In this paper, GPU optimization techniques focusing on the depletion solver are presented. The optimization focuses on two aspects: enhancing the performance of the matrix solver and reducing the overheads of the CPU operations. The performance of the optimized depletion solver will be demonstrated thereafter.

2. Optimization of Matrix Exponential Solver

2.1 Non-Zero Element Major (NZEM) Storage

Conventional CPU-based solvers prefer to configure the array of the systems with region major (RM) storage. RM arrays are not only readable and user-friendly, but also favorable for the caching mechanism of CPUs where the temporal locality of access is important. Due to these benefits, the prototype GPU depletion solver had adopted the RM storage scheme.

However, as demonstrated on **Figure 2**, NZEM storage is more suitable for coalesced memory accesses on GPUs when region-wise parallelism is applied. Under the RM storage scheme, the accesses are strided and the memory coalescing is barely expectable.

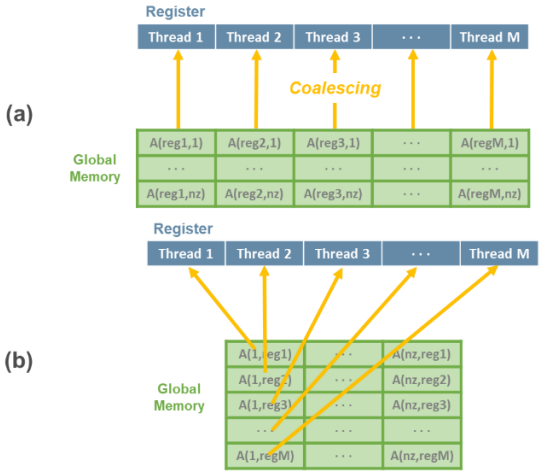

Figure 2 Global memory access patterns under (a) NZEM and (b) RM storage schemes.

Nevertheless, the systems are setup from the CPU side, and in this aspect, RM storage has a benefit that it does not require the transposition of systems for the use in the GPU solver. To utilize NZEM storage, the CPU needs to transpose the RM array to the NZEM array, while if the RM storage is utilized on GPU, the transposition process is not required.

2.2 Separating Diagonal and Off-diagonal Elements

The prototype setup and solver routines were storing the arrays in complex double types; it is because that Chebyshev rational approximate method (CRAM) is defined in the complex space. However, not all the elements are complex; while the vectors are all complex numbers, only the diagonal elements of the matrices are complex numbers as shown in equation (1). Therefore, it is possible to separate the matrices into diagonal and off-diagonal parts and use different data types; the former uses complex double while the latter uses native double.

$$\mathbf{n} = \alpha_0 \mathbf{n}_0 + \mathrm{Re}\left(\sum_j \alpha_j \left(\mathbf{A}\Delta t - \theta_j \mathbf{I} \right)^{-1} \mathbf{n}_0 \right) \qquad (1)$$

where

$\mathbf{n}$	Number density
α_j	j-th residue (complex)
θ_j	j-th pole (complex)
$\mathbf{A}$	Burnup matrix

This separation is beneficial in terms of memory usage and access burden of off-diagonal elements. Considering that the off-diagonal part takes 80% of a matrix, it can reduce the memory usage of a unit matrix significantly and thereby allowing to increase the size of each batch. In addition, it becomes much easier to locate the diagonal elements in a matrix. Sparse matrix formats all have its own mapping arrays to locate elements, and nTRACER also utilizes the compressed sparse row (CSR) format to save the burnup matrices. If diagonal elements are included in the non-zero array with off-diagonals, access to the diagonal terms should be done by an indirection using the mapping arrays. On the other hand, elements in a separate diagonal array can be directly accessed.

2.3 Gauss-Seidel Iterative Solver

The GPU depletion solver has employed biconjugate gradient stabilized method (BiCGSTAB) for the matrix inversion. However, it turned out that the Gauss-Seidel (GS) method is more efficient for inverting the matrices in CRAM due to large diagonal dominance. As the result, GS could also converge in a few iterations and eventually BiCGSTAB required more operations than GS. Thus, we decided to replace the iterative linear system solution method.

Also, GS can reduce the size of buffer memories. Due to the region-wise parallelism, each thread should have separate buffers, named workspace, to save intermediate vectors like the residual or the solution of the previous iteration. A thread requires seven temporary vectors with BiCGSTAB, while only three are required in GS.

To simply put, GS is cheaper than BiCGSTAB as the increase in the number of iterations is small enough to be compensated by the reduction of operations, and it also uses smaller memory due to the reduced workspace size.

3. Overhead Optimization

3.1 Implicit Transposition during Setup

It was pointed out that to use NZEM on the solver, an explicit transposition of the systems is required. To avoid this, an implicit transposition, which simply generates the systems directly in the NZEM storage from the setup phase was implemented. This increases the system setup time as the NZEM storage is not CPU-friendly, but the transposition burden can be eliminated.

Figure 3 illustrates the CPU cache usage of NZEM and RM storages. When an NZEM array is written, the cache block does not contain the other elements in a region. On the other hands, the cache with RM array holds the data of a region which the thread takes. This increase the use rate of cache and, consequently, ensure the better parallel efficiency.

This eliminates the temporary copy of the systems which involves a rearrangement of arrays from RM to NZEM. If the CPU transposes the systems explicitly, the systems first have to be copied to a different buffer with reversed storage before copying the systems to the GPU. This temporary copy time on CPU used to take so much time, therefore, the goal of the implicit transposition is to eliminate the overhead of creating a temporary copy of the systems on CPU in spite of the increased setup cost.

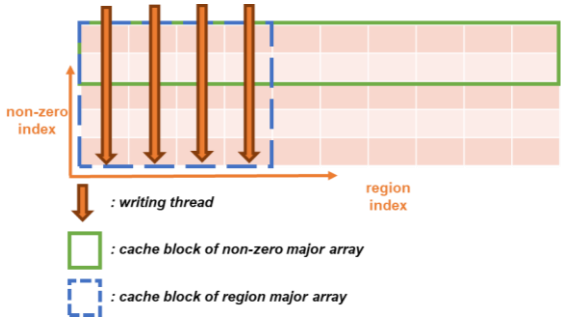

Figure 3 Cache usage of CPU cores with different storages.

3.2 Fast Non-zero Index Search

This is to reduce the additional burden caused by the implicit transposition. Since the depletion system setup is performed on CPU, it is critical to the performance of 3D calculation as the CPU resources get limited [2]. Thus, unlike other tasks, this focuses on the efficiency of CPU operations.

When locating the non-zero elements, both setup and solver routines utilize the CSR mapping arrays. The main problem of using the mapping arrays is that an iteration is required to locate a specific element. While this is not a problem for the solver which sweeps all the elements sequentially, the setup routine has to jump between the elements to generate the systems.

Therefore, a separate non-zero index table is defined which receives a reaction type and a target nuclide and returns the corresponding non-zero index. The pseudo-codes in **Figure 4** below illustrate how the pre-defined index table can be effectively used to find the non-zero indices. *ix* and *ir* indicates the index of the target nuclide and the reaction type, respectively. The arrays, *isoRx* and *inzRx*, stores the indices of isotope and non-zero element after neutron reaction.

```
INTEGER FUNCTION idxsrch(ix,ir)
  INTEGER :: ix, ir, i
  DO i = Rowptr(ir), Rowptr(ir+1)-1
    IF (colidx(i) .EQ. isoRx(ix,ir)) EXIT
  END DO
  idxsrch = i
END FUNCTION
```

(a)

```
INTEGER FUNCTION idxsrch(ix,ir)
  INTEGER :: ix, ir
  idxsrch = inzRx(ix,ir)
END FUNCTION
```

(b)

Figure 4 Pseudo-codes for non-zero index search with (a) linear search and (b) pre-defined index map.

3.3 Explicit Copy with CUDA API

PGI Fortran supports the intrinsic assignment operator (=) to copy arrays between CPU and GPU. However, it turned out that the operator is in fact substituted by a pre-defined function during the compile time which was not efficient. Hence, the assignment operators were replaced with the *cudaMemcpy* API, and all the copy operations now fully exploit the PCI-e bandwidth.

4. Performance Analysis

This section presents the performance analysis results for the APR1400 2D core problem [5], and the core was depleted up to 15 GWd/tHM. The time specified in the graphs are the total computing time in the cycle depletion calculation. The computing resources used are listed in **Table 1**. The CPU solver used the Intel Fortran compiler while the GPU solver was compiled with PGI Fortran.

Table 1 Specification of the workstation.

CPU	2 × Intel Xeon E5-2630 v4 20 Cores, 2.4 GHz (Boost)
GPU	NVIDIA GeForce RTX 2080 Ti
Compiler	PGI Fortran 19.4 Intel Fortran 19.0.4

The base performance of the GPU solver without any optimizations and the performance of the CPU solver are compared at **Figure 5**. 'Sys. Setup' indicates the time for the system setup, 'Sol' is the time for the CRAM solution, and 'Copy' means all the data copy involving the systems; namely, the data copy between CPU and GPU and the temporary copy on the CPU side. Finally, 'Post' is the time for post-processes like updating number densities with the solutions of CRAM.

The solution time shows slight improvement, but the other parts were poorer. Due to the inferior performance of the PGI compiler, system setup takes longer time than CPU. Furthermore, the copy time is literally zero in the CPU solver as it does not involve communications with GPU, while it occupies a significant portion in the GPU solver. As the result, the GPU-based depletion solver has become even slower than the CPU calculation in total. Such result motivated this research to reduce the burden of ancillary overheads in the GPU solver.

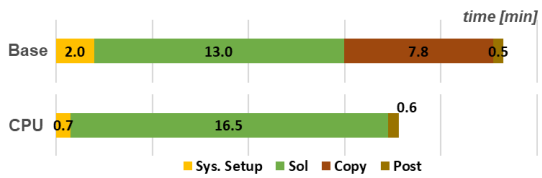

Figure 5 Depletion performance comparison between the base case and the CPU solver.

4.1 Enhancements of Matrix Exponential Solver

The progressive improvement of the performance of the solver by the optimizations is illustrated in **Figure 6**. From the comparison of the RM case and NZEM case, it can be noted that 25% of the time was reduced. This large reduction is due to a better memory access coalescing.

Combined with NZEM scheme, matrix separation had reduced the time by 30%. The major factor is the reduced overheads to access off-diagonal elements. Direct access to diagonal elements may affect less than the former, due to fewer number of diagonals.

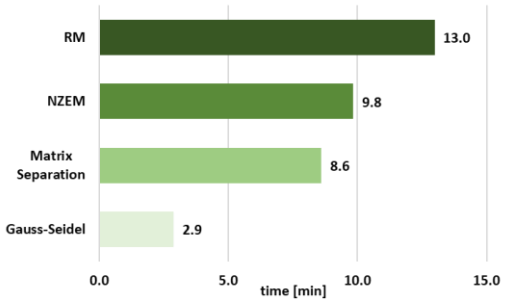

Figure 6 Matrix exponential solution time comparison.

However, among all the optimizations, applying GS to CRAM is the most effective treatment. It is no wonder that it achieves the greatest improvement as operations and memory accesses were reduced significantly.

4.2 Reduction of the Overheads

The progressive reduction of the overheads is shown in **Figure 7**. In this case, 20 cores of CPU were used for all the calculations.

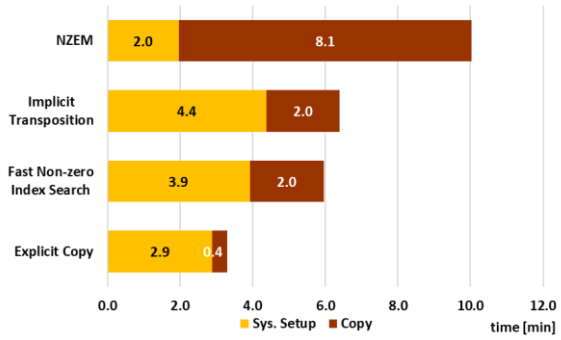

Figure 7 Depletion solver CPU overhead reduction.

By adopting the implicit transposition, the majority of the copy overhead had disappeared. This optimization eliminated both the transposition and the temporary copy on CPU. As the result, the copy time was reduced to one-fourth. Using the CUDA memory copy APIs reduced the copy time further by another 75%, and as the result, 95% of the copy overhead had been eliminated.

Nevertheless, the system setup time increased due to the worsened cache utilization on CPU. While this is an expected consequence, it is an unexpected result that the

fast non-zero index search using the pre-defined index map shows little effect.

Next investigation is the dependence of the overheads with the number of CPU cores. Recall from our previous research [2] that nTRACER employs a plane-per-GPU distributed parallel topology and a typical GPU cluster mounts multiple GPUs per node. Resultantly, the number of CPU cores that can be assigned per GPU is decreased in 3D calculations. Therefore, the behavior of overheads with reduced number of CPU cores should be examined to confirm the scalability to 3D calculations.

Figure 8 demonstrates the profile for the two different usages of CPU cores. The digits, '5' and '20', specify the number of used CPU cores.

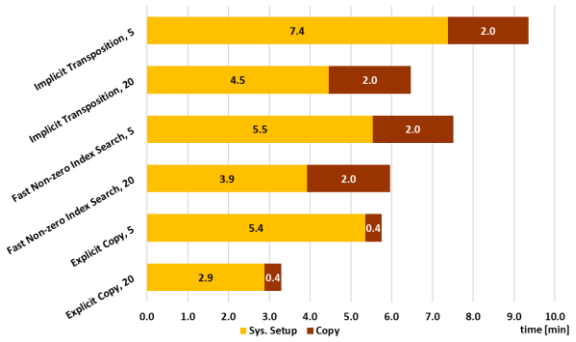

Figure 8 Overheads comparison by different usage of CPUs.

Only comparing the 20-core cases, fast index search seems less effective with only 30 seconds of reduction. However, it becomes powerful with limited use of CPU resources with 2 minutes shorter time. Therefore, in the aspects of the system setup performance, the fast search optimization takes an important role for 3D calculations.

4.3 Overall Performance of GPU Depletion Solver

At the early in this chapter, the poorer performance of GPU depletion solver was pointed out. However, after a few progresses, the GPU solver outperforms the one of CPU version. And, the results are described on **Figure 9**. '5' and '20' stands for the usage of CPU resources, and 'Base' case is from the initial version and 'Optimized' case is from the latest version after various optimizations.

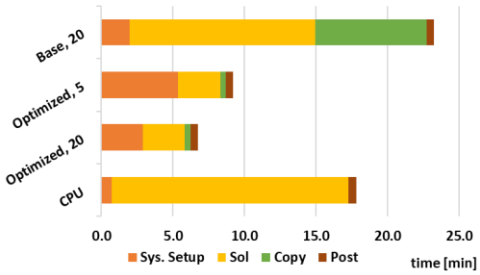

Figure 9 Performance of Various Depletion Solvers

Even with the full CPU resources, the GPU solver had not overcome the CPU solver except the solution time. Several times of optimizations lessen the incidental overheads and had shortened the solution time. With some sacrifice of setup, the total depletion calculation can be finished within a few minutes.

5. Conclusion

nTRACER had completed GPU acceleration of all the hotspot procedures in cycle the depletion. However, the GPU depletion calculations had little improvement over the CPU solver. Therefore, additional optimizations for the GPU depletion solver have been made in this work. Specifically, calculating the matrix exponentials using the iterative CRAM solver and unnecessary overheads by CPUs were optimized.

NZEM storage system, separation of burnup matrices and Gauss-Seidel as a linear system solver were adopted to shorten the solution time. Also, implicit transposition, fast non-zero index search during setup and explicit copy through CUDA API functions makes the overheads less burdensome. Some of them causes adverse effects, but overall performance of the depletion solver becomes way better.

Nevertheless, the enlarged portion of system setup as a result of implicit transposition is still an issue. It seems small enough when using maximal CPU resources, but it takes over 50 % of the whole time of depletion. One excuse for this big portion is the absolute time is still much less than matrix exponential solution time of the CPU solver. Considering all the limitations, the GPU solver is much better due to the fact that same degradation at the CPU solver is expected to occur with limited number of cores.

ACKNOWLEDGEMENTS

This research is supported by National Research Foundation of Korea (NRF) Grant No. 2016M3C4A7952631 (Realization of Massive Parallel High Fidelity Virtual Reactor)

REFERENCES

[1] N. Choi, J. Kang and H. G. Joo, "Preliminary Performance Assessment of GPU Acceleration Module in nTRACER," Transactions of the Korean Nuclear Society Autumn Meeting, Yeosu, Korea, Oct. 24-25, 2018.

[2] H. G. Lee, S. Jae, N. Choi, J. Kang and H. G. Joo, "Progress of GPU Acceleration Module in nTRACER for Cycle Depletion," Jeju, Korea, July. 9-10, 2020.

[3] M. Pusa, "Rational Approximations to the Matrix Exponential in Burnup Calculations," Nuclear Science and Engineering 169(2), pp. 155-167, 2011.

[4] A.Yamamoto, M. Tatusmi and N. Sugimura, "Numerical Solution of Stiff Burnup Equation with Short Half Lived Nuclides by the Krylov Subspace Method," Journal of Nuclear Science and Technology 44(2), pp. 147-154, 2007.

[5] H. Hong and H. G. Joo, "Analysis of the APR1400 PWR Initial Core with the nTRACER Direct Whole Core Calculation Code and the McCARD Monte Carlo Code," Transactions of the Korean Nuclear Society Spring Meeting, Jeju, Korea, May 18-19, 2017.