Transactions of the Korean Nuclear Society Virtual Spring Meeting
July 9-10, 2020

Applicability of GPU Acceleration for RAST-K Fast Reactor Depletion Solver

Siarhei Dzianisau, Deokjung Lee*
Department of Nuclear Engineering, Ulsan National Institute of Science and Technology, 50 UNIST-gil, Ulsan, 44919
*Corresponding author: *deokjung@unist.ac.kr*

1. Introduction

Fuel depletion calculations are one of the most valuable calculations in reactor core design that modern computer simulation codes can do. They are widely used in such applications as new reactor design and multi-cycle fuel loading pattern (LP) search for existing nuclear power plants (NPP). Given such an important role, depletion solvers of modern codes are expected to be fast and accurate at the same time.

A commonly used method that meets both of stated requirements is Chebyshev Rational Approximation Method (CRAM) applied to a system of burnup equations as discussed in [1]. This method allows finding individual nuclide number densities as functions of fuel burnup using numerically determined fitting coefficients of a chosen order. Being a reliable technique, CRAM is used in our inhouse nodal diffusion code RAST-K [2].

Currently, RAST-K can perform fuel depletion calculations for thermal reactors (TR) with square geometry fuel assembly (FA) lattice. However, development of a fast reactor (FR) module with hexagonal FA geometry for RAST-K is on its final stage. One of the key requirements for the new module is fast computation speed. Therefore, it was decided to develop both CPU and graphics card (GPU) versions of the code to check for potential benefits of using GPU acceleration in FR code.

2. RAST-K Fast Reactor Depletion Solver

RAST-K is originally developed for a single-core CPU simulation. Hence, it can be used on any personal computer (PC). Since some of user PCs already have an installed GPU, it is attractive to provide a built-in code capability of using GPU acceleration. Another reason for introducing GPU acceleration specifically in FR module is its' presumably larger computation time compared to TR module. In more details, a comparison of FR and TR codes is discussed in subsection 2.1.

2.1 Comparison of Thermal Reactor and Fast Reactor Depletion Solvers

The difference between TR and FR modules is caused by the difference in neutron spectrum for each of these modules. Thus, RAST-K TR code is using only 2 neutron energy groups whereas a new FR code has to use 24 neutron groups. Fast neutrons are known to have relatively small interaction cross-sections with the entire range of nuclides unlike thermal neutrons that not only have generally higher values of cross-sections but also higher discrepancy of those values for different nuclides. As a result, TR module can produce an accurate result while updating only a small group of most important nuclides such as fuel nuclides and fission products (FP) with high values of absorption cross-sections. On the contrary, FR module needs to account for a larger portion of nuclides since more heavy metal nuclides (HM) are fissile in fast neutron spectrum, and most of fission products have their cross-section values in approximately the same range.

Another point worth mentioning is a slight difference in geometry. Most TR in the world are using rectangular FA geometry, which is divided plain-wise into 4 geometrical nodes. As for FR, they are usually designed using a hexagonal geometry, which requires using 6-node plain-wise division in the way discussed in [3]. All in all, those factors make the FR code more computationally intensive as it needs to account for higher number of variables in previously mentioned burnup calculations. This yields a necessity to accelerate the code using alternative approaches such as running part of the code on GPU.

2.2 GPU Acceleration

As shown by Pusa, the formula for updating the list of nuclide number densities $\boldsymbol{N}$ is the following:

$$\boldsymbol{N} = \alpha_0 \boldsymbol{N_0} + 2 \cdot \boldsymbol{real}\left(\sum_{j=1}^{\frac{order}{2}} \alpha_j \left(\boldsymbol{A}t - \theta_j \boldsymbol{I}\right)^{-1} \boldsymbol{N_0}\right) \quad (1)$$

In equation (1), matrix and vector components are stated in bold capital letters, other components are either real or complex numbers. In terms of GPU acceleration (omitting operations of multiplication by a number for brevity), this formula can be restated as shown on Fig. 1.

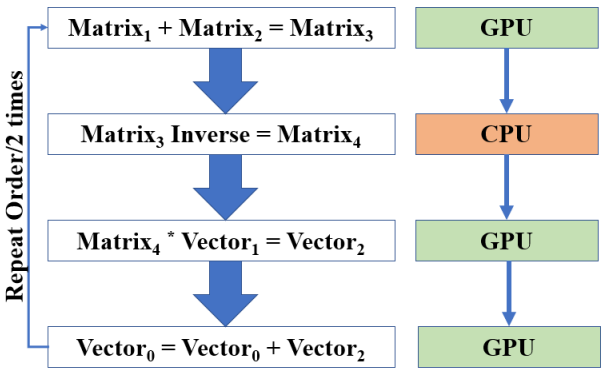

Fig. 1. GPU acceleration algorithm for CRAM.

As pointed out on Fig. 1, almost all matrix operations of the method can be effectively accelerated on GPU

except for matrix inverse. The original RAST-K code is using Gauss-Jordan method for matrix inverse calculation. This method is not applicable for parallel computation on GPU as it requires a strict sequence of operations. Therefore, it should be computed on CPU because a single CPU core is much more powerful than a single GPU core (though, the advantage of a GPU is that it consists of thousands of cores). As for other matrix inverse methods that would be suitable for highly parallel computation on GPU, no currently available effective methods were found in literature at the moment of writing this paper. Nonetheless, even a partial acceleration of the code could be beneficial for the overall code performance.

2.3 Speed Comparison Results

For the speed comparison of a CPU and a GPU version of RAST-K FR Depletion Solver, the following initial conditions and equipment were used. RAST-K is written in Fortran language. The GPU acceleration routines were written using CUDA Fortran [4]. Therefore, most of the code stayed the same for both CPU and GPU versions, which is advantageous in terms of code conversion from CPU to GPU version. A single core of Intel Core i7 7700K at 4,37GHz was used for running both versions of the code, and nVidia GeForce RTX 2060 SUPER was used for GPU acceleration. For the GPU version of the code, an algorithm shown on Fig. 1 was implemented.

The test problem of choice was a single node fuel depletion calculation for 221 nuclides (28 HM and 193 FP) and 22 burnup points. The result of calculation could be found in Table I.

Table I: Comparison of CPU and GPU versions of RAST-K Fast Reactor Depletion Solver

Parameter	Pure CPU	CPU+GPU	Difference, %
Time per burnup step, sec	0.22	0.18	18.18
Total time, sec	4.48	3.98	11.16

The following observations could be made based on the result shown in Table I. First, even adding a small portion of GPU accelerated code could noticeably increase the speed of computation, which is expected to save a proportional amount of computation time when applied to a real core consisting of thousands of geometrical nodes. Second, the result of the GPU accelerated code includes operations of copying data from CPU to GPU and back at each burnup step, and additionally for each matrix inverse calculation in the middle of each step. As stated in [4], operations of copying data to and from GPU are quite time-consuming and should be avoided as much as possible. Hence, further improvements of the code by converting a larger part of the CPU code into GPU code could result in further speed improvements.

The most challenging part of adding GPU acceleration to RAST-K FR Depletion Solver is to find an effective method for parallel matrix inverse calculation. This would allow to eliminate data copying in the middle of each burnup step thus reducing the time of computation.

Finally, important observation could be made based on the data type used in this study and the difference between the CPU and the GPU calculation results for the last 22nd burnup point. Since double precision is used for both CPU and GPU codes, no difference in result was observed based on straight-forward subtraction of the CPU result from the GPU result for all studied burnup points. However, GPUs may benefit from using single precision as it would allow storing twice as much data on their limited on-board memory as well as further increase the speed of computation.

3. Conclusions

A potential application of GPU acceleration for RAST-K Fast Reactor Depletion Solver was evaluated in this study. It was found that running even a small portion of matrix operations on a GPU could noticeably reduce required computation time. At the same time, some matrix operations such as finding a matrix inverse are not currently applicable for GPU acceleration. Therefore, future studies are expected on finding an effective highly parallel method for inverse matrix determination.

A detailed analysis of the studied test problem showed that excessive copying of data to and from GPU could cause a noticeable drop in the overall performance. Hence, future work on converting a larger part of the code into GPU code is planned in order to further improve the computation speed.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT). (No.NRF-2017M2A8A2018595)

REFERENCES

[1] M. Pusa, Rational Approximations to the Matrix Exponential in Burnup Calculations, Nuclear Science and Engineering, Vol.169, No.2, pp.155-167, 2011.
[2] J. Choe, S. Choi, P. Zhang, J. Park, W. Kim, H. Shin, H. Lee, J. Jung, D. Lee, Verification and validation of STREAM/RAST-K for PWR analysis, Nuclear Engineering and Technology, Vol.51, No.2, pp.356-368, 2019.
[3] J. Y. Cho, B. O. Cho, H. G. Joo, S. Q. Zee, S. Y. Park, Non-linear triangle-based polynomial expansion nodal method for hexagonal core analysis, KAERI/TR--1652/2000, 2000.
[4] M. Fatica, G. Ruetsch, CUDA Fortran for Scientists and Engineers, Morgan Kaufmann, 2014.