GPU ACCELERATION OF DOPPLER BROADENING FOR NEUTRON TRANSPORT CALCULATIONS[1]

Paul E. Burke[1], Kyle E. Remley[2], and David P. Griesheimer[2]

[1]Georgia Institute of Technology
770 State St. NW, Atlanta, GA 30313

[2]Naval Nuclear Laboratory
PO Box 1072, Schenectady, NY 12301

paul.burke@gatech.edu, kyle.remley@unnpp.gov, david.griesheimer@unnpp.gov

ABSTRACT

In radiation transport calculations, the effects of material temperature on neutron/nucleus interactions must be taken into account through Doppler broadening adjustments to the microscopic cross section data. Historically, Monte Carlo transport simulations have accounted for this temperature dependence by interpolating among precalculated Doppler broadened cross sections at a variety of temperatures. More recently, there has been much interest in on-the-fly Doppler broadening methods, where reference data is broadened on-demand during particle transport to any temperature. Unfortunately, Doppler broadening operations are expensive on traditional central processing unit (CPU) architectures, making on-the-fly Doppler broadening unaffordable without approximations or complex data pre-processing. This work considers the use of graphics processing unit (GPU)s, which excel at parallel data processing, for on-the-fly Doppler broadening in continuous-energy Monte Carlo simulations. Two methods are considered for the broadening operations – a GPU implementation of the standard SIGMA1 algorithm and a novel vectorized algorithm that leverages the convolution properties of the broadening operation in an attempt to expose additional parallelism. Numerical results demonstrate that similar cross section lookup throughput is obtained for on-the-fly broadening on a GPU as cross section lookup throughput with precomputed data on a CPU, implying that offloading Doppler broadening operations to a GPU may enable on-the-fly temperature treatment of cross sections without a noticeable reduction in cross section processing performance in Monte Carlo transport codes.

KEYWORDS: Doppler Broadening, GPU, Transport Methods

1. INTRODUCTION

Neutron transport algorithms require accurate representation of the fundamental underlying cross section data. An important aspect of this is accurate modeling of the impact of material temperature on cross sections due to the thermal motion of the target nuclei during particle transport. This effect is modeled via

[1]This manuscript has been authored by Fluor Marine Propulsion under contract No. DOE-89233018CNR000004 with the U.S. Department of Energy. The United States government retains and the publisher, by accepting this article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, and world-wide license to publish, distribute, translate, duplicate, exhibit, and perform the published form of this manuscript, or allow others to do so, for United States Government purposes.

Doppler broadening, which has the effect of "broadening" sharp cross section resonances into smoother, wider peaks as material temperature increases. Traditionally, effects of material temperature have been handled in neutron transport calculations by pre-calculating broadened cross sections at various temperatures representative of the problem being solved and interpolating the cross section on this temperature mesh. These broadened cross sections are most frequently calculated using the SIGMA1 method [1–2]. While the broadening itself is exact, the interpolation between temperature points carries an amount of error, depending on the fineness of this mesh [3–4]. Additionally, storing tabulated cross sections at many temperatures results in increasingly large memory requirements.

An alternative to pre-computed cross sections is on-the-fly Doppler broadening. With on-the-fly broadening, cross section data is only stored at the coldest temperature in the problem, and the broadening operation is performed at the time of cross section lookup. With this on-the-fly scheme, the cross section data is exact to the degree of the specific broadening operation used, and only one set of cross sections needs to be stored (at the coldest temperature). However, the operations involved in broadening are typically expensive. Other methods have been discussed for on-the-fly broadening which are less computationally expensive, including the windowed multipole method [5] and a regression-based on-the-fly method [6].

In this paper, we explore the possibility of using general-purpose graphics processing units (GPGPUs or just GPUs) to perform on-the-fly Doppler broadening computations. GPUs excel at problems that can be expressed as a set of massively many smaller tasks. This reflects the underlying architecture in the device itself. Whereas traditional central processing units (CPUs) have the hardware capability to operate with up to tens of threads at a time, GPUs have hardware to operate with tens of thousands of threads at a time. It is thus of interest to investigate whether the operations involved with Doppler broadening can be expressed in this way. This paper presents two Doppler broadening algorithms that were implemented on a GPU. The first is an implementation of the standard SIGMA1 method [1] optimized for GPU computation. The second is a novel method that relies on the fact that the Doppler broadening kernel used in SIGMA1 computation is fundamentally a convolution and leverages an existing fast convolution method in an attempt to expose additional parallelism in the operation, thus making it more suitable for computation on a GPU.

2. THE SIGMA1 DOPPLER BROADENING ALGORITHM

We begin by briefly reviewing the SIGMA1 algorithm. The analytical form of the Doppler broadening equations for a cross section at broadened temperature T_b from [1] is

$$\begin{aligned} \sigma(y, T_\mathrm{b}) &= \sigma^*(y, T_\mathrm{b}) - \sigma^*(-y, T_\mathrm{b}) \,, \\ \sigma^*(y, T_\mathrm{b}) &\equiv \frac{1}{y^2\sqrt{\pi}} \int_0^\infty x^2 \sigma(x, T_\mathrm{ref}) \exp(-(x-y)^2)\, dx \,, \end{aligned} \tag{1}$$

where the variable transformations

$$y = \sqrt{\alpha E}\,, x = \sqrt{\alpha E_r}\,, \alpha = \frac{M}{k_b(T_\mathrm{b} - T_\mathrm{ref})} \tag{2}$$

have been applied. The standard method implemented in this study is the SIGMA1 method [1] as implemented in NJOY [2]. This method operates on linearly-interpolable tabulated cross sections at reference temperature T_ref

$$\sigma(E, T_\mathrm{ref}) = \left(\frac{E - E_k}{E_{k+1} - E_k}\right)\sigma_{k+1} + \left(\frac{E_{k+1} - E}{E_{k+1} - E_k}\right)\sigma_k : \; E \in [E_k, E_{k+1}] \,. \tag{3}$$

Using the change of variables in Eq. (2), Eq. (3) becomes

$$\sigma(y, T_{\mathrm{ref}}) = \sigma_i + s_i\left(x^2 - x_i^2\right):\ x \in [x_i, x_{i+1}]\,,$$
$$s_i = {(\sigma_{i+1} - \sigma_i)}\Big/{\left(x_{i+1}^2 - x_i^2\right)}\,. \tag{4}$$

With this notation, the Eq. (1) can be expressed as (from [2])

$$\sigma^*(y) = \sum_{i=1}^{N}\left\{J_i\left[\sigma_i - s_i x_i^2\right] + K_i s_i\right\}, \tag{5}$$

where the index variable i spans all cross section data points in the energy range [y – 4, y + 4] and the J_i, K_i terms can be expressed as

$$J_i = \frac{1}{y^2}H_{2,i} + \frac{2}{y}H_{1,i} + H_{0,i}\,,$$
$$K_i = \frac{1}{y^2}H_{4,i} + \frac{4}{y}H_{3,i} + 6H_{2,i} + 4yH_{1,i} + y^2H_{0,i}\,, \tag{6}$$

and $H_{n,i}$ are given as

$$H_{n,i} = F_n(x_i - y) - F_n(x_{i+1} - y)\,,$$
$$F_0(a) = \frac{1}{2}\mathrm{erfc}(a)\,,$$
$$F_1(a) = \frac{1}{2\sqrt{\pi}}\exp(-a^2)\,, \tag{7}$$
$$F_n(a) = \frac{n-1}{2}F_{n-2}(a) + a^{n-1}F_1(a)\,.$$

A recent demonstration mini-app [7] proposed a slight re-ordering of operations to enable compiler auto-vectorization, which effectively groups together calculations of the F_n functions:

$$\begin{aligned}\sigma^*(y) = {} & J_1^*[\sigma_1 - s_1x_1^2] + K_1^*s_1 \\ & + \sum_{i=2}^{N-1}\left\{-\left(J_i^*\left[\sigma_{i-1} - s_{i-1}x_{i-1}^2\right] + K_i^*s_{i-1}\right) + \left(J_i^*\left[\sigma_i - s_ix_i^2\right] + K_i^*s_i\right)\right\} \\ & - \left(J_N^*[\sigma_{N-1} - s_{N-1}x_{N-1}^2] + K_N^*s_{N-1}\right),\end{aligned} \tag{8}$$

where J_i^*, K_i^* are expressed as

$$J_i^* = \frac{1}{y^2}F_2(x_i - y) + \frac{2}{y}F_1(x_i - y) + F_0(x_i - y)\,,$$
$$K_i^* = \frac{1}{y^2}F_4(x_i - y) + \frac{4}{y}F_3(x_i - y) + 6F_2(x_i - y) + 4yF_1(x_i - y) + y^2F_0(x_i - y)\,. \tag{9}$$

This re-ordered algorithm is used for the implementation described in this paper.

3. NOVEL VECTORIZED METHOD OVERVIEW

The SIGMA1 method (and its re-ordering) is parallelizable to the degree that there is an independent set of operations for each reference cross section data point in the relevant data range. By leveraging a fast convolution method [8], an additional level of parallelism can be exposed, breaking each set into a smaller collections of independent operations. First, we observe (like some other modern methods, as in [9]) that the Doppler broadening equation (Eq. (1)) can be cast as a convolution operation,

$$\begin{aligned} \sigma^*(y, T_\mathrm{b}) &= \frac{1}{y^2\sqrt{\pi}} \Sigma(x; T_\mathrm{b}, T_\mathrm{ref}) * G(x) \,, \\ \Sigma(x; T_\mathrm{b}, T_\mathrm{ref}) &\equiv x^2 \sigma(x, T_\mathrm{ref}) \,, \\ G(x) &\equiv e^{-x^2}. \end{aligned} \tag{10}$$

When expressed in this form, it is possible to leverage certain properties of convolution integrals. For instance, Reference [8] describes how the convolution of two piecewise polynomial functions can be quickly computed via a conversion to "delta function integral representation" – an alternative way of representing such functions. This technique can be used if both $\Sigma(x; T_\mathrm{b}, T_\mathrm{ref})$ and $G(x)$ can be cast as piecewise polynomials. Cross section data are already given in piecewise linear form, which becomes piecewise quadratic under the transform given by Eq. (2):

$$\begin{aligned} \sigma(E, T_\mathrm{ref}) &= A_k + B_k E \;:\; E \in [E_k, E_{k+1}] \,, \\ A_k &= \frac{E_{k+1}\sigma_k - E_k \sigma_{k+1}}{E_{k+1} - E_k}, B_k = \frac{\sigma_{k+1} - \sigma_k}{E_{k+1} - E_k} \,, \\ \Sigma(x; T_\mathrm{b}, T_\mathrm{ref}) &= A_k x^2 + \frac{1}{\alpha} B_k x^4 \;:\; x \in [x_k, x_{k+1}] \,. \end{aligned} \tag{11}$$

Reference [8] describes how piecewise polynomials like this can be converted to an equivalent delta function representation. This single representation is valid over the entire domain of the function, and aids in the computation of the convolution. Performing this conversion yields the representation of $\Sigma(x; T_\mathrm{b}, T_\mathrm{ref})$:

$$\begin{aligned} \Sigma(x; T_\mathrm{b}, T_\mathrm{ref}) = \sum_{j=1}^{N} \Bigg\{ & \begin{bmatrix} x_j^2 \{ (A_j - A_{j-1}) + E_j (B_j - B_{j-1}) \} \\ x_j \\ -1 \end{bmatrix} \\ & + \begin{bmatrix} x_j \{ 2(A_j - A_{j-1}) + 4E_j (B_j - B_{j-1}) \} \\ x_j \\ -2 \end{bmatrix} + \begin{bmatrix} 2(A_j - A_{j-1}) + 12E_j (B_j - B_{j-1}) \\ x_j \\ -3 \end{bmatrix} \\ & + \begin{bmatrix} \frac{24}{x_j} E_j (B_j - B_{j-1}) \\ x_j \\ -4 \end{bmatrix} + \begin{bmatrix} \frac{24}{x_j^2} E_j (B_j - B_{j-1}) \\ x_j \\ -5 \end{bmatrix} \Bigg\}, \end{aligned} \tag{12}$$

where, from [8], the triplet notation is defined

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = X_1 * \delta^{-X_3}(x - X_2) \,, \tag{13}$$

$$\delta^{-m}(t-t_i)=\begin{Bmatrix}0 & t<t_i\\ \dfrac{(t-t_i)^{m-1}}{(m-1)!} & t\geq t_i\end{Bmatrix}.$$

Recalling the definitions of A_j and B_j from Eq. (11) it is readily apparent that

$$\begin{aligned}A_k+E_kB_k=A_{k-1}+E_kB_{k-1}=\sigma\left(E_k,T_{ref}\right)\\ \Leftrightarrow (A_k-A_{k-1})=-E_k(B_k-B_{k-1})\,.\end{aligned}\tag{14}$$

Also, note that when $\sigma\left(E_k,T_{ref}\right)=0$,

$$\begin{aligned}A_k+E_kB_k=\sigma\left(E_k,T_{ref}\right)=0\\ \Leftrightarrow A_k=-E_kB_k\,.\end{aligned}\tag{15}$$

From this, Eq. (12) reduces to

$$\begin{aligned}\Sigma(x;T_{\mathrm{b}},T_{\mathrm{ref}})=\sum_{j=1}^{N}\Bigg\{&\begin{bmatrix}x_j\{-2(A_j-A_{j-1})\}\\ x_j\\ -2\end{bmatrix}+\begin{bmatrix}-10(A_j-A_{j-1})\\ x_j\\ -3\end{bmatrix}+\begin{bmatrix}-\dfrac{24}{x_j}\,(A_j-A_{j-1})\\ x_j\\ -4\end{bmatrix}\\ &+\begin{bmatrix}-\dfrac{24}{x_j^2}\,(A_j-A_{j-1})\\ x_j\\ -5\end{bmatrix}\Bigg\}.\end{aligned}\tag{16}$$

In order to apply the method given in [8], $G(x)$ needs also to be expressed as a piecewise polynomial. A piecewise step decomposition that preserves the integral between any two mesh points was used:

$$\begin{aligned}G(x)=\ C_k\ :\ x\in[p_k,p_{k+1}]\,,\\ C_k=\frac{\int_{p_k}^{p_{k+1}}e^{-x^2}dx}{p_{k+1}-p_k}=\frac{\frac{1}{2}\sqrt{\pi}(\mathrm{erf}(p_{k+1})-\mathrm{erf}(p_k))}{p_{k+1}-p_k}\ .\end{aligned}\tag{17}$$

Transforming Eq. (17) to delta function integral representation yields

$$G(x)\ =\sum_{i=1}^{P}\left\{\begin{bmatrix}(C_i-C_{i-1})\\ p_i\\ -1\end{bmatrix}\right\}.\tag{18}$$

With Eqs. (16) and (18), the convolution can be computed analytically by using the identity

$$\begin{bmatrix}A_1\\ A_2\\ A_3\end{bmatrix}*\begin{bmatrix}B_1\\ B_2\\ B_3\end{bmatrix}=\begin{bmatrix}A_1B_1\\ A_2+B_2\\ A_3+B_3\end{bmatrix}.\tag{19}$$

The resultant convolution of $\Sigma\left(x;T_B,T_{ref}\right)$ and $G(x)$ is then

$$\begin{aligned}\Sigma(x;T_{\mathrm{b}},T_{\mathrm{ref}})*\ G(x)=\sum_{j=1}^{N}\sum_{i=1}^{P}\left\{\begin{bmatrix}-2x_j\Psi_j\Gamma_i\\ x_j+p_i\\ -3\end{bmatrix}+\begin{bmatrix}-10\Psi_j\Gamma_i\\ x_j+p_i\\ -4\end{bmatrix}+\begin{bmatrix}-\dfrac{24}{x_j}\Psi_j\Gamma_i\\ x_j+p_i\\ -5\end{bmatrix}+\begin{bmatrix}-\dfrac{24}{x_j^2}\,\Psi_j\Gamma_i\\ x_j+p_i\\ -6\end{bmatrix}\right\},\\ \Gamma_i=(C_i-C_{i-1})\,,\\ \Psi_j=\left(A_j-A_{j-1}\right).\end{aligned}\tag{20}$$

The domain of this convolution is (0,∞]. However, in practice, it is preferable to truncate the convolution evaluation to a smaller region of interest (typically from $y - 4$ to $y + 4$). Outside of this truncation range, the cross section is set to zero. With this, the vectorized Doppler broadening method proceeds as follows:

Pre-calculation – can be performed at time of cross section data import

1. Convert (energy, cross section) data point pairs to the Ψ_j form in Eqs. (11) and (20) and store. These data points are independent of temperature and can be used in a broadening operation to any target temperature greater than the reference temperature.
2. Select and store a set of P mesh points and construct and store the Γ_i data as described in Eqs. (17) and (20). In order to match up with the ($y - 4$ to y + 4) importance region from which the cross section is evaluated, mesh points only need span the region [–4, 4].

Calculation at time of lookup

1. Calculate the constant α used in the change of variables to x-space.
2. Take a subset of cross section data points that includes the energy range [$y - 4$, $y + 4$].
3. Calculate A_0, $A_1 - A_0$ using a prepended data point (0.99 * E_0, 0 barns), where E_0 is the energy of the lowest energy cross section data point in the region of interest. The rest of the data subset is appended. The net result of this subset construction is a delta function integral representation of a fictitious cross section that is identical to the real cross section within the energy range of [$y - 4$, $y + 4$] and artificially set to 0 below the region of interest. It is this subset (say of N points) that is iterated over in the first summation term in Eq. (20).
4. Calculate the $4 \times N \times P$ individual terms of the double summation in Eq. (20). Note that each term can be calculated in parallel, and the contribution contains only simple floating point operations, as opposed to function evaluations.
5. If necessary, repeat step 4 for the $\sigma^*(-y, T_b)$ term. This need only be computed when $y < 4$.
6. Plug the result of the convolution into Eq. (10).

4. GPU IMPLEMENTATION DETAILS

This study implemented both the SIGMA1 method and novel vectorized methods as CUDA kernels. All cross section data was read and stored in global memory on the GPU. The lookup parameters (i.e., lookup energy, temperature, etc.) are assembled into buffers on the CPU, then sent to the GPU at the time of kernel dispatch. In order to make the overhead associated with data transfer and kernel launch small with respect to total runtime, lookups were batched and dispatched all at the same time. In order to hide data transfer latency, these batches were further broken up into inner batches, to allow data transfer to happen concurrently with computation.

4.1. SIGMA1 Implementation

The CUDA implementation of the SIGMA1 broadening operation leverages the fact that each iteration of the summation in Eq. (8) can be computed in parallel and that the local results can be reduced at the end. However, typical CPU implementations of SIGMA1 use a method to start in the middle of the data subset (the subset of the cross section data that spans [$y - 4$, $y + 4$]) and traverse to the edges of the set, ensuring that the method uses only the data that is necessary. This kind of traversal poses a problem for a parallelized implementation, where *a priori* knowledge of the bounds of our dataset before computation is desired. Furthermore, searches for specific energy points in a cross section evaluation are comparatively slow relative to the speed of the broadening operation. Therefore, the SIGMA1 implementation searches for an approximate data subset, which is guaranteed to at least contain the required cross section data, using a lethargy mapping function.

4.2. Novel Vectorized Implementation

The CUDA implementation of the novel vectorized operation adds another level of parallelism to the broadening operation—parallelism in P-space (i.e., along the various P mesh points of the $G(x)$ decomposition). This allows for more threads to work on the same lookup. Similar logic as discussed in §4.1 applies to the searches for the bounds of the data subset.

5. TIMING RESULTS

To gauge the performance of utilizing GPUs for Doppler broadening, an exploratory mini-app was created with both CPU and GPU implementations of the traditional SIGMA1 and the new vectorized Doppler broadening algorithms. The objective of the mini-app was to assess cross section lookup throughput, in lookups per second, or LUP/s, as a function of implementation and lookup batch size. Comparisons of the two GPU implementations with respect to cross section lookups without Doppler broadening on a CPU, which is the standard practice in most Monte Carlo codes, as well as CPU SIGMA1 on-the-fly Doppler broadening, which represents the "worst possible case" of a performance hit when adding on-the-fly broadening to a transport simulation were of particular interest.

All implementations were run through the mini-app on a compute node with an 18-core (36-thread) Intel Xeon Gold 6140 Processor and 2 NVIDIA Tesla v100 GPUs (only 1 of which was used). This exercise compared raw throughput (in LUP/s) as a function of batch size, in lookups. It was expected that overall throughput is expected to increase as the number of lookups dispatched at once increases, due to the presence of fixed overhead costs. To investigate this behavior, lookup throughput was measured for various batch sizes for lookups in the resolved resonance range of the ENDF/B-VII.0 evaluation of U-238. These results are shown in Figure 1. Also plotted in Figure 1 is the CPU SIGMA1 throughput as well as the reference lookup throughput, which is the performance of cross section lookups without on-the-fly Doppler broadening.

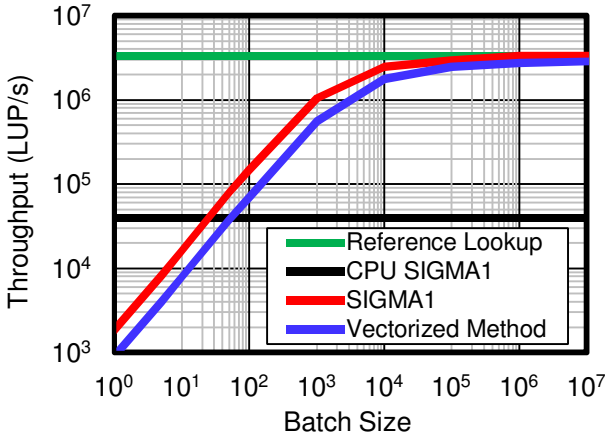

Figure 1. Throughput (in LUP/s) of cross section lookups for U-238 as a function of batch size for the various cross section lookup methodologies implemented.

As seen in Figure 1, throughput increases for both the GPU SIGMA1 and vectorized method implementations as batch size increases, as expected, up to a batch size of around 10^4. As batch size approaches 10^7, both GPU on-the-fly Doppler broadening methods have throughput that matches or exceeds the reference lookup. This suggests that similar computational performance can be achieved with on-the-fly Doppler broadening on GPUs as can be achieved through CPU cross section lookups without on-the-fly Doppler broadening. Figure 1 also indicates that the GPU SIGMA1 implementation outperforms the

vectorized method. This is a counterintuitive result since the vectorized method is *prima facie* "more parallel" than SIGMA1. However, relative gains in parallelism with the vectorized method appear to be offset by the larger computational costs imposed by the method. In addition, some latencies in dispatching SIGMA1 broadening were hidden through batching GPU kernel dispatches, which can potentially expose further parallelism in SIGMA1 broadening.

6. SUMMARY AND CONCLUSIONS

This paper presents GPU implementations for the SIGMA1 Doppler broadening method as well as a novel vectorized algorithm that leverages special properties of the convolution integral. An exploratory mini-app was used to compare the computational performance of these implementations against CPU cross section lookups without Doppler broadening and a CPU SIGMA1 implementation. Numerical results from the mini-app demonstrate that GPU implementations of Doppler broadening outperform CPU implementations if cross section lookups are batched together prior to GPU dispatch. In fact, for large dispatch batch sizes, the Doppler broadening throughput on a GPU is similar to the CPU throughput for cross section lookups without on-the-fly Doppler broadening. A direct comparison of results for the two GPU broadening methods reveals that the SIGMA1 method consistently outperforms the newly-developed vectorized algorithm. Although the SIGMA1 algorithm is theoretically "less parallel" than the vectorized algorithm, and hypothetically less suitable for implementation on a GPU, this algorithmic parallel inefficiency can be effectively overcome by dispatching many SIGMA1 operations concurrently on the GPU. The results of this study demonstrate that Doppler broadening algorithms are well-suited for implementation on GPUs. Furthermore, preliminary results generated with a representative Monte Carlo mini-app suggest that offloading broadening operations to a GPU may enable on-the-fly Doppler broadening during Monte Carlo transport simulations without a significant decrease in performance relative to static temperature interpolation methods used in many contemporary codes.

REFERENCES

1. D.E. Cullen and C.R. Weisbin, "Exact Doppler Broadening of Tabulated Cross Sections," *Nucl. Sci. Eng.*, **60**, 199-229 (1976).
2. R.E. MacFarlane and D.W. Muir, "The NJOY Nuclear Data Processing System Version 91," LA-12740-M (Oct. 1994)
3. T. Trumbull, "Treatment of Nuclear Data for Transport Problems Containing Detailed Temperature Distributions," *Nucl. Tech.*, **156**, 75-86 (2006).
4. S. Peng, A.B.S. Zhang, X. Jiang, "Investigations on the Point-wise Neutron Cross-section Temperature Interpolation Methods," *Annals of Nuclear Energy*, **45**, 155-160 (2012).
5. C. Josey et al., "Windowed Multipole for Cross Section Doppler Broadening", *Journal of Computational Physics*, **307**, 715-727 (2016).
6. W.R. Martin, S. Wilderman, F.B. Brown, G. Yesilyurt, "Implementation of On-the-Fly Doppler Broadening in MCNP," *Proc. Of International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Technology (M&C 2013)*, Sun Valley, ID (2013).
7. E. Brun, S. Chauveau, F. Malvagi, "PATMOS: A prototype Monte Carlo transport code to test high performance architectures" *M&C 2017*, Jeju, Korea (2017).
8. R. J. Polge and R. D. Hays, "Numerical Technique for the Convolution of Piecewise Polynomial Functions", *IEEE Transactions on Computers*, **C-22 (11)**, 970-975 (1973).
9. G. Ferran, W. Haeck, and M. Gonin, "A New Method for the Doppler Broadening of the Solbrig's Kernel Using a Fourier Transform", *Nuclear Science and Engineering*, **179:3**, 285-301 (2015).