INL/CON-10-17677
PREPRINT

Using GPU Programming for Inverse Spectroscopy

INMM 2010

David Gerts
Hugh Wimberly
Nathaniel Fredette

July 2010

The INL is a
U.S. Department of Energy
National Laboratory
operated by
Battelle Energy Alliance

Using GPU Programming for Inverse Spectroscopy

By Dr. David Gerts, Hugh Wimberly, Nathaniel Fredette
Idaho National Laboratory
P.O. Box 1625, Idaho Falls, ID 83415

Abstract:

The Idaho National Laboratory (INL) has developed a detector that relies heavily on computationally expensive inverse spectroscopy algorithms to determine probabilistic three dimensional mappings of the source and its intensity. This inverse spectroscopy algorithm applies to material accountability by determining the location and strength of nuclear materials. Because of the computational expense, the INL has incorporated new hardware from the commercial graphics community. General programming for graphics processing units (GPUs) is not a new concept. However, the application of GPUs to evidence theory-based inverse spectroscopy is both novel and particularly apropos. Improvements while using a (slightly upgraded) standard PC are approximately three orders of magnitude, making a ten hour computation in less than four seconds. This significantly changes the concept of prohibitively expensive calculations and makes application to materials accountability possible in near real time, with the limiting condition reduced to sensor signal acquisition time.

Introduction

Inverse spectroscopy has application in a wide variety of nuclear materials and nuclear non-proliferation applications. The confirmation that a specific isotope is present in an unknown sample is an example of inverse spectroscopy. Furthermore, determining the intensity and location of an unknown radioactive sample is a more dramatic example of inverse spectroscopy. In order to address these types of known inverse spectroscopy applications in nuclear materials management, Idaho National Laboratory (INL) developed a novel algorithm to perform inverse spectroscopy.

INL's algorithm is based on using the electron energy, photon interaction site, and electron direction in the recreation of the 3D location and intensity of an unknown source. We expect, however, that all of the parameters based on the electron path will have significant uncertainties. A major question remains of how to use the information in the presence of significant uncertainty. Our answer is to use the Dempster-Shafer Theory of Evidence to combine the evidence of each electron path for a final data fusion solution.

Review of Traditional Inverse Spectroscopy Methods

A current area of interest in the field of nuclear nonproliferation concerns the ability to infer the identity and position of a radiation source through the analysis of a measured gamma spectrum output from a detection system. Historically, this spectrum analysis has been conducted through various peak-fitting techniques. These existing methods are sufficient to

identify most radiation sources, but they both possess a significant shortcoming when applied to current detection problems. Peak fitting methods only generate functions that approximate the measured data and do not use any additional information about the source. Because of this, these methods cannot calculate the source position.

Various function approximation techniques have been employed to fit the measured output data of gamma detectors. An example is shown in Figure 1.

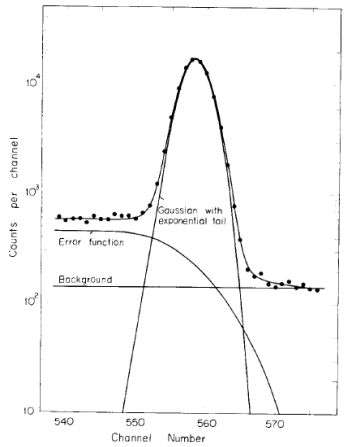

Figure 1: Example of fitting gamma spectrum with a function [1]

One of the simplest of these methods uses the principle of least squares to approximate the measured gamma spectra. In [2],Salmon uses a multichannel analyzer with *n* channels numbered 1… *i*… *n* to collect a gamma spectrum from a mixture of *m* nuclides numbered 1… *j*… *m*. Based on this system, a relationship between the count rate a_{ij} in channel *i* from nuclide *j* and the count rate from mixture b_i can be determined [1].

$$b_i = \sum_{j=1}^{m} a_{ij} x_j + Z_i, \qquad \{1\},$$

where x_j is the ratio of the unknown activity to a corresponding standard activity and Z_i is a random error. Finally, the method of least squares is applied where the random error Z_i is minimized and a set of *m* equations numbered 1… *k*… *m*:

$$\sum_{j=1}^{m} x_j \sum_{j=1}^{n} a_{ik} a_{ij} = \sum_{j=1}^{n} a_{ik} b_i. \qquad \{2\}$$

This system of equations then can be written in the form of $Ax = y$ and can be solved by $x = A^{-1} y$.

In addition to the method of least squares, Gaussian-plus-exponential approximation has been used to recreate gamma spectra. In this approximation, the Gaussian and exponential tail shape was chosen because a typical gamma spectrum is expected to have a Gaussian count distribution around a photopeak energy with significantly more counts at energies lower than the photopeak energy due to scattering events. This method uses a linear approximation to model background radiation and is employed in the widely used gamma spectrum analysis tool SAMPO80 [3]. Because this Gaussian-plus-exponential approximation is only valid in the intervals of distinct photopeaks, search routines must be used to determine these intervals in the measured spectra. Once these intervals have been determined a normalized least squares fitting routine is used to generate shape calibration lines that determine the width and height of the Gaussian distribution and the junction points between the Gaussian, exponential and background functions. This method is summarized by Equations {3} and {4}.

$$\chi^2 = \sum_{i=k-l}^{k+m} \frac{\left(n_i - f_i\right)^2}{n_i}, \qquad \{3\},$$

where *i* is the channel number, n_i is the number of counts in channel *i*, *k* is the approximate center channel of the peak and *l* and *m* are the channels specifying the fitting interval. The function f_i has the following values seen in Equation (4) based on the value of *i*

$$f_i = p_1 + p_2\left(i - p_4\right) + p_3 e^{-\frac{1}{2}\frac{\left(i-p_4\right)^2}{p_5^2}}, \text{ for } p_4 - p_6^2 \le i \le p_4 + p_7^2$$

$$f_i = p_1 + p_2\left(i - p_4\right) + p_3 e^{-\frac{1}{2}p_6^2\frac{\left(2i-2p_4+p_6^2\right)}{p_5^2}}, \text{ for } i < p_4 + p_7^2 \qquad \{4\},$$

$$f_i = p_1 + p_2\left(i - p_4\right) + p_3 e^{-\frac{1}{2}p_7^2\frac{\left(2p_4-2i+p_7^2\right)^2}{p_5^2}}, \text{ for } i > p_4 + p_7^2,$$

where p_1 is the constant in the continuum approximation, p_2 is the slope in the continuum approximation, p_3 is the height of the Gaussian, p_4 is the centroid of the Gaussian, p_5 is the width of the Gaussian, p_6^2 is the distance in channels to the lower junction point between the background and exponential functions and p_7^2 is the distance in channels to the higher junction point between the Gaussian and exponential functions [4]. Finally, one more routine is used to determine the height of the Gaussian distributions and the coefficients of the polynomial approximation of the continuum. The routine uses a least squares technique:

$$\chi^2 = \sum_{i=k-l}^{k+m} \frac{\left(n_i - b_i - \sum_{j=1}^{np} f_{ij}\right)^2}{n_i}, \qquad \{5\},$$

where *i* is the channel number, n_i is the number of counts in channel *i*, *k* is the reference channel for the background polynomial, *l* and *m* are the channels specifying the fitting interval, b_i is the background function $b_i = p_1 + p_2\left(i-k\right) + p_3\left(i-k\right)^2$, and np is the number of peaks in the interval [3]. The function f_i in equation {5} has the following values:

$$f_{ij} = p_{2+2j} e^{-\frac{1}{2}\frac{\left(i-p_{3+2j}\right)^2}{w_j^2}}, \text{ for } p_{3+2j} - l_j \le i \le p_{3+2j} + h_j$$

$$f_{ij} = p_{2+2j} e^{-\frac{1}{2}l_j\frac{\left(2i-2p_{3+2j}+l_j\right)}{w_j^2}}, \text{ for } i < p_{3+2j} - l_j \qquad \{6\},$$

$$f_{ij} = p_{2+2j} e^{-\frac{1}{2}h_j\frac{\left(2p_{3+2j}-2i+h_j\right)}{w_j^2}}, \text{ for } i > p_{3+2j} + h_j$$

where p_1, p_2, and p_3 define the continuum; p_{2+2j} and p_{3+2j} define the height and centroid of the j^{th} peak; and w_j, l_j and h_j define the line shape by linearly interpolating the shape calibration results.

Inverse Spectroscopy Using Electron Paths

Compton scatter dominates the cross section in supersaturated air for energies of 100 keV to 10 MeV, as shown in Figure 2. Because this encompasses nearly the entire gamma range for fission fragments, it is sufficient to incorporate only Compton scatter in a proof of principle algorithm for photons. Although the Compton relation and the associated Klein-Nishina formula are standard equations in neutral particle transport, they are rarely cast in a form that

emphasizes the interacting electron. These equations are solved for the electron scatter angle and the electron energy. The electron energy and path are the observable quantities. The photon to electron scatter angle is the independent variable and the initial photon energy is calculable from these two parameters. The scattering geometry is shown in Figure 3 below.

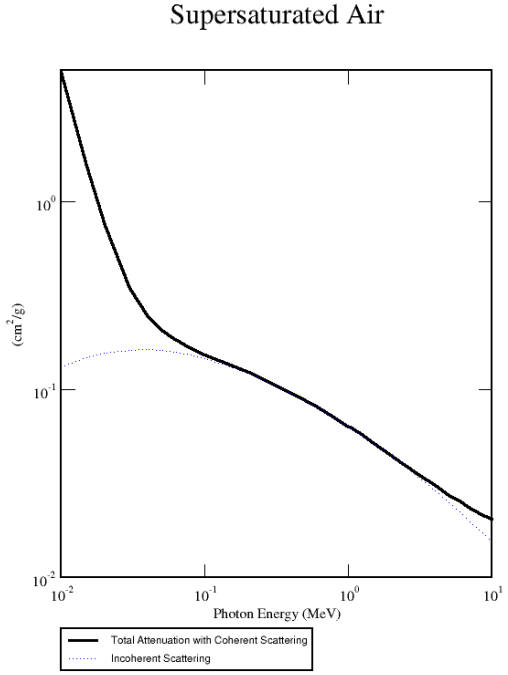

Figure 2: Photon cross section for supersaturated air (~105% humidity at 30°C) [5]

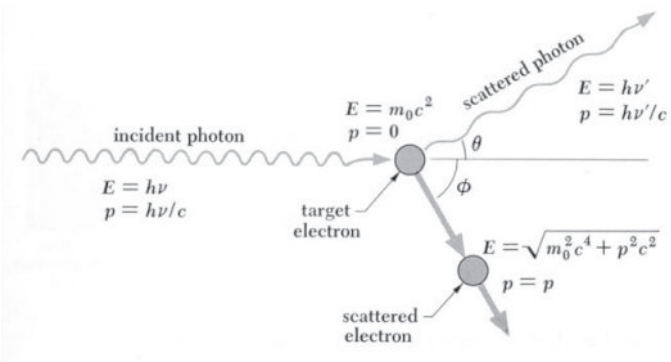

Figure 3: Compton scatter geometry [6]

The classic Compton relation for the energy of the electron, E_{elec}, given the scattering angle, j , of the electron and the initial photon energy, E_o, is

$$E_{elec} = E_o \frac{2\,{}^{E_o}\!/_{511keV}\, Cos^2(\varphi)}{\left(1+{}^{E_o}\!/_{511keV}\right)^2 - \left({}^{E_o}\!/_{511keV}\right)^2 Cos^2(\varphi)}. \qquad \{7\}$$

The unpolarized, double differential in angle Klein-Nishina formula for the electron scatter angle based on the incident photon energy, scattered photon energy E ¢, and the photon scatter angle, q , is

$$\frac{d\sigma_{elec}}{d\Omega} = \frac{r_o^2}{2}\left(\frac{E'}{E_o}\right)^2\left(\frac{E_o}{E'}+\frac{E'}{E_o}-Sin^2(\theta)\right)\left(-\frac{4\left(1+{}^{E_o}\!/_{511keV}\right)^2 Cot(\varphi)Csc^3(\varphi)}{\left[\left(1+{}^{E_o}\!/_{511keV}\right)^2+Cot^2(\varphi)\right]^2}\right), \quad \{8\}$$

where r_o is the classical electron radius. The first portion of the differential cross section is the familiar Klein-Nishina formula for photon scatter. The last parentheses, however, accounts for the electron scatter angle.

This formulation for the Klein-Nishina formula is not in a useable form for solving the inverse problem. Several more relations are needed. The relation between the photon scatter angle and the electron scatter angle is

$$\theta = 2\,ArcTan\left(\frac{Cot(\varphi)}{1+{}^{E_o}\!/_{511keV}}\right). \quad \{9\}$$

The relation between the incident photon energy, the scatter electron energy, and the scattered electron energy is

$$E' = E_o - E_{elec}. \quad \{10\}$$

Equations {7} through {10} can be solved for the doubly differential cross section given the electron scatter energy in terms of the electron scatter angle, j . The double differential cross section plotted in Figure 3 shows the electron scatter angle versus the interaction probability. The plot assumes that the electron energy was measured as 200 keV. In this case, the most probable angle between the electron and the incident photon was near zero degrees and has a maximum scatter angle of 66.136 degrees.

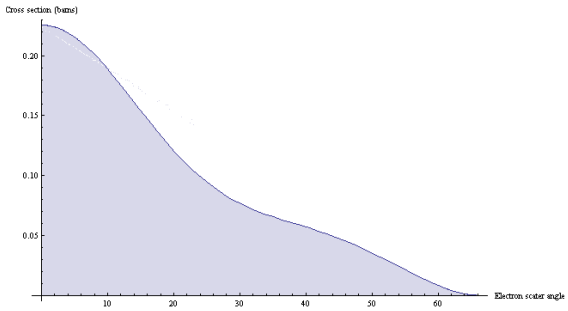

Figure 4: Double differential cross section for electron scatter angle given 200 keV electron

The maximum scatter angle can be found by the equation

$$\varphi_{max} = ArcCos\left(\frac{22.3607E_{elec}}{\sqrt{511+500E_{elec}}}\right), \quad \{11\}$$

where j is in radians. Having a maximum scatter angle is simply the result of applying conservation of energy and momentum to the scattering problem.

The result of all of these calculations is a 3D mapping of probability of where the photon may have originally arisen. This can be shown as in Figure 5 where the green cloud represents probability of the original gamma source location.

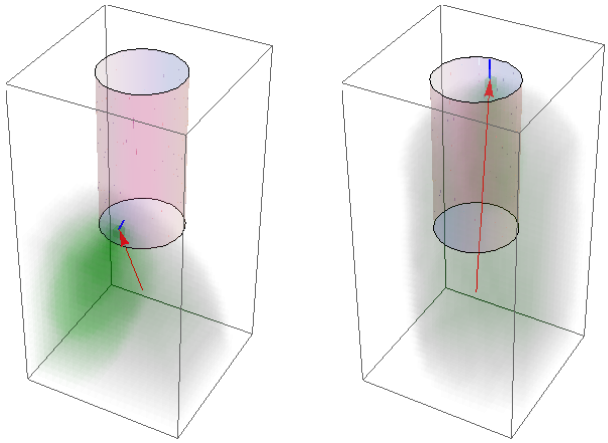

Figure 5: Probability maps for two sample electron tracks.

This is clearly not the final solution because it has failed to account for background, scattered radiation from the source, and the uncertainty. In addition, the solution is clearly not unique. In fact, the focus of this research is *not* the calculation of the probability mappings, but rather, the addition of the probability mappings for many, many particle tracks.

Dempster-Shafer Theory of Evidence

Dempster-Shafer (DS) theory begins with a "question of interest."[7] It is important that this question be sufficiently broad. DS theory is based on the use of belief functions. In comparison, a probability function is additive, whereas a belief function is not, in general. The key to the concept of belief functions is the limited division of belief. Whereas probability functions assume that belief is apportioned to the points in the frame, belief functions allow basic probability numbers to be assigned to whole sets of points in the frame of discernment without further subdivision. The basic ideas is that a whole belief is divided into one or more basic probability numbers, *b*(A) and allocated to one or more subsets A, called focal elements such that

$$\sum\left\{b(A)\middle|A\subseteq\Theta\right\}=1. \qquad \{12\}$$

The basic probability number *b*(A) allocated to a focal element A is not further divided into smaller chunks allocated to proper subsets of A. These basic probability numbers are related to belief functions by

$$Bel(A)=\sum\left\{b(B)\middle|B\subseteq A\right\}, \qquad \{13\}$$

where B represents proper subsets of A and the summation is over all sets, B.

Given two pieces of evidence with basic probability numbers assigned to them, DS theory of belief functions combines the two (or more) pieces of evidence with the symbol, $\breve{A}$,using Dempster's orthogonal rule of combination [8],

$$b=b_1\oplus b_2$$
$$b(A_i)=\frac{\sum_{A_p\cap A_q=A_i}b_1(A_p)b_2(A_q)}{1-\sum_{A_p\cap A_q=\varnothing}b_1(A_p)b_2(A_q)}. \qquad \{14\}$$

Given these calculus mechanics, it is possible to combine any number of pieces of evidence in support of a particular answer within the frame of discernment (noting that answers can be combined into supersets within the frame of discernment).

In short, using the Dempster-Shafer theory of evidence for sensor fusion reduces to a problem of determining basic probability numbers for the data to be evaluated, and then, interpreting the results. It is appropriate for an application that fundamentally relies on a random distribution of photons to arrive at a detector to specify the interaction density.

Graphics Processing Unit Background

Graphics processing units (GPUs) have traditinally been very limited, since they were designed for a very specific application. However, as the demands of computer gaming have skyrocketed, graphics cards have become increasingly capable. In recent years, due to the specialized nature of graphics processing, the raw throughput potential of GPUs has come to dramatically overshadow CPUs of a similar cost.

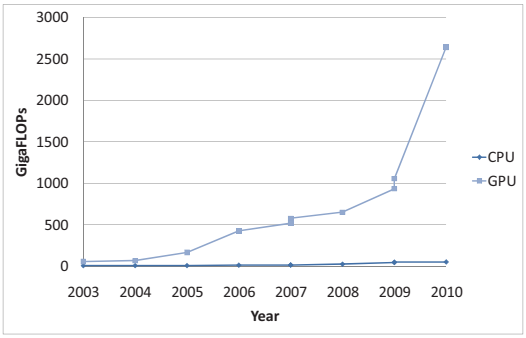

Figure 6: Computational power cost equivalent between GPU to CPU

GPUs would already have replaced CPUs if not for the fact that GPUs are only designed to operate over a limited problem domain. Although they are incredibly fast, they are fundamentally designed to operate on pixels, colors, spatial vectors and positions. In this domain, integers are used more heavily than floating-point numbers, addition and multiplication are much more important than division or trancendental functions, and numerical precision is second to speed. Most importantly, GPUs are designed to operate on large numbers of independent data items, so they are heavily parallized and heavily pipelined.

Due to increased interest in using the cheap-but-specialized GPUs for compute-intense projects and programs, NVIDIA and ATI have started increasing the capability of their graphic cards in non-traditional roles. The most significant hardware change has been to improve floating-point performance, especially for double-precision floating-point numbers, but the most significant change has been to make APIs and compilers available for programmers to use to more easily access the functionality of GPUs for general-purpose computing.

For our project, we used an NVIDIA GeForce GTX 285, a card that is no longer in production, but retailed for about $300 at the time of our use. We used NVIDIA's Compute Unified Device Architecture (CUDA) [9], a set of extensions to C that expose both low- and high-level operations on NVIDIA GPUs. Programming with CUDA primarialy requires an understanding of how CUDA parallelizes problems.

GPU Rationale

GPUs have many inherent characteristics that make them ideal for certain types of problems:

1. Obvious parallel algorithm – algorithms that do not rely on sequential processing
2. Compute intensive – problems that have a high arithmetic to input/output operation ratio

3. Lack of data interdependence – data elements can be operated upon without reading many other data elements
4. Data locality – data elements that sequentially organized in memory
5. Modest memory requirements – in 2010, this implies entire sub-problems that can fit within 1.5 gigabytes of memory
6. Small data types – can use integers or single-precision floating-point numbers
7. High degree of Fused-Multiple Add Operations – uses addition and multiplication, rather than division and transcendental functions

CUDA Programming Concepts

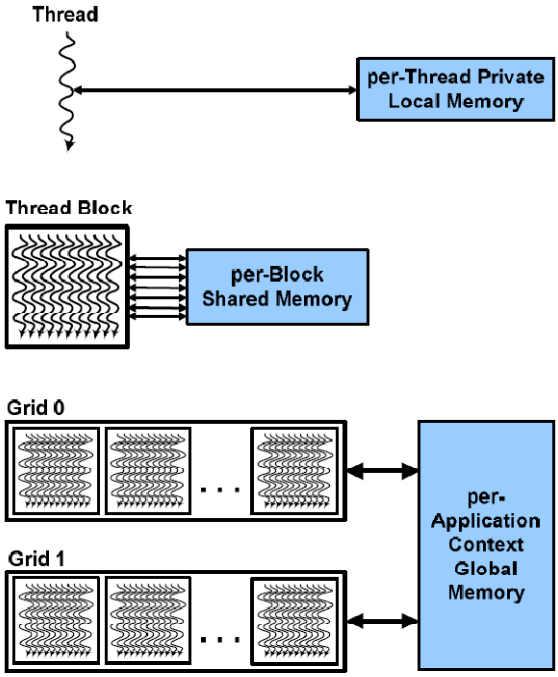

Figure 7: Diagram of CUDA Programming Architecture [9]

Code running on the CPU, called host code, calls a *kernel* function on blocks of data. Each kernel call is divided up into any number of *blocks* which can be exectuted in any order. Each block consists of up to 512 threads, grouped into *warps* no larger than 32 threads each. All the threads in a warp can potentially execute at the same time; as long as they perform the same operations, they remain in sync. As soon as one or more threads in a warp follows a branch that other threads do not, the different groups are serialized and one will be executed until completion before the remainder of the warp is executed. Thus, the ideal programming senario is one where conditional branches can be mostly or entirely avoided.

Statement of the Problem

Our approach is to use Dempster's Rule of Combination, given in equation {14}, on the probability maps from the electron tracks. Dempster's Rule of Combination, applied to each spatial cell, requires $O(N^3)$ operations. However, each of those operations requires foreknowledge of K, the measure of conflict. K only needs to be calculated once for each group of combinations, but requires finding the summation of $O(N^6)$ products. In the case of 100x100x100 cells, there are easily one trillion calculations (multiple and adds, primarily).

However, Dempster's rule of combination is trivial to parallel because the rows are independent of each other and can thus be calculated separately. Therefore, it makes for an ideal application on a graphics processing unit (GPU).

Using GPUs for Dempster's Rule of Combination

By far, the most expensive portion of combining two probability maps is computing the measure of conflict for the denominator. This could be represented in C++ as the very simple nested loop:

```
float compute_coefficient_host(float *m1, float *m2) {
      float product_sum = 0;
      for (unsigned int i; i < MATRIX_SIZE; i++)
         for (unsigned int j = 0; j < MATRIX_SIZE; j++)
               if (i != j)
                     product_sum += m1[i]*m2[j];
      return 1 / (1 - product_sum);
}
```

This computation is just an extensive multiply-accumulate, the kind of operation that is perfectly ideal for optimizing into FMAD (floating-point multiply-add) or FFMA (floating-point fused multiply-add) operations on a GPU. In CUDA, instead of calling the above computation, a CUDA kernel function would be called as below:

```
compute_coefficient_cuda<<<NUM_BLOCKS, NUM_THREADS>>>(m1, m2, &result);
```

To cause `NUM_BLOCKS` blocks, each with `NUM_THREADS` threads, to each compute the sum of a portion of the final coefficient. The GPU will start scheduling blocks in any order until it runs out of threads to allocate. On a modern GPU, this will generally result in 340-512 threads operating simultaneously. Since FMADs have a theoretical throughput of 8 per clock cycle per multiprocessor our theoretical speed for this function is 8 ops/cycle · multiprocessor × 30 multiprocessors × 1476 MHz, or more than 350 GFLOPs.

Results

Among all the problem sets we tested ($10 \le N \le 100$), we were able to maximize efficiency by using the maximum number of threads, 512, and having each thread multiply and sum 512 pairs of numbers. After each such iteration, we put the numbers into an array which was summed and put into a "higher-order" array recursively, so that we required only $O(\log_{512}(N^6))$ additional memory. In addition to increasing computational efficiency, this striding helped preserve numerical stability.

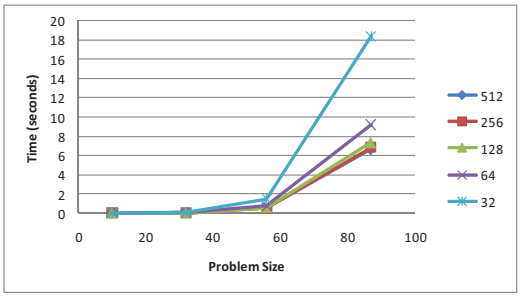

Figure 8: Threadcount differences

The overhead for bussing data to and from the GPU on our machine was approximately 60ms; for N<40, this time dominated the time required to get a solution. For our smallest test set, N=10 and both the CPU and GPU computed the combination of two probability maps in less

than a tenth of a second, with the GPU taking approximately four times as long. For all larger test sets, the time required by the CPU was much less than required by the GPU. We achieved our goal of three orders of magnitude speed-up for sufficiently large data sets: our 100x100x100 test set took more than 1000 times as long to compute on the CPU as on the GPU.

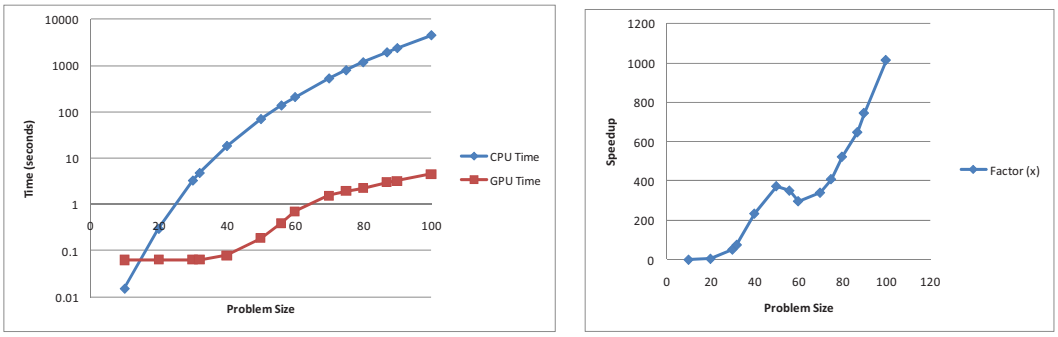

Figure 9: Comparison of computational cost for Dempster's rule of combination

Conclusions

General programming for GPU processors is simple and stable enough to use without great technical expertise, and inexpensive enough to be accessible by anyone. For applications that fit the execution model of GPUs, performance gains can be enormous, and worth the still-great expense of redesigning the most costly portion of the code to execute in a CUDA kernel.

References

1. Dojo, M., *SHAPE FUNCTION OF PHOTOPEAKS FOR GAMMA-RAY SPECTRUM ANALYSIS WITH GE(LI) DETECTORS.* Nuclear Instruments & Methods, 1974. **115**(2): p. 425-429.
2. Salmon, L., *ANALYSIS OF GAMMA-RAY SCINTILLATION SPECTRA BY THE METHOD OF LEAST SQUARES.* Nuclear Instruments & Methods, 1961. **14**(2): p. 193-199.
3. Koskelo, M.J., P.A. Aarnio, and J.T. Routti, *SAMPO80 - MINICOMPUTER PROGRAM FOR GAMMA-SPECTRUM ANALYSIS WITH NUCLIDE IDENTIFICATION.* Computer Physics Communications, 1981. **24**(1): p. 11-35.
4. Routti, J.T. and S.G. Prussin, *PHOTOPEAK METHOD FOR COMPUTER ANALYSIS OF GAMMA-RAY SPECTRA FROM SEMICONDUCTOR DETECTORS.* Nuclear Instruments & Methods, 1969. **72**(2): p. 125-&.
5. X-5_Monte_Carlo_Team, *MCNP -- A General Monte Carlo N-Particle Transport Code, Version 5.* 2003.
6. Evans, R.D., *The atomic nucleus*. International series in pure and applied physics. 1955, New York,: McGraw-Hill. 972 p.
7. Yager, R.R. and L. Liu, *Classic works of the Dempster-Shafer theory of belief functions*. Studies in fuzziness and soft computing. 2008, Berlin ; New York: Springer. xix, 806 p.
8. Dempster, A.P., *The Dempster-Shafer calculus for statisticians.* International Journal of Approximate Reasoning, 2008. **48**(2): p. 365-377.
9. *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. 2009; Available from: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.