ABSTRACT

CALDERÓN JR., ADAN FAUSTINO. Forward Monte Carlo Calculation of Coincidence Gamma-Ray Spectra. (Under the direction of Robin P. Gardner.)

The detector response functions were generated with Monte Carlo and used as the libaries that were fit to experimental gamma-ray spectra using a least squares approach. A code named MCNP-CP was built and used for comparison

Forward Monte Carlo Calculation of Coincidence Gamma-Ray Spectra

by
Adan Faustino Calderón Jr.

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Nuclear Engineering

Raleigh, North Carolina

2014

APPROVED BY:

John K. Mattingly	Jason Osborne

Robin P. Gardner
Chair of Advisory Committee

DEDICATION

For Adalyn Fayth Calderón.

BIOGRAPHY

Adan Calderón was born in Laredo, TX on September 6, 1979 to Adan F. Calderón Sr. and Ada Linda Calderón. He grew up in Laredo and went through its public school system. In 1998 Adan enrolled at Texas A&M University where he later received a Bachelor of Science Degree in Nuclear Engineering and a minor in Mathematics. Through the years his passion has always been working on and with computers, electronics and networks. His early computing cluster building skills comes from where it was necessary to decode passwords based of off one way hash functions. He is a licensed amateur radio operator and has interests in wireless communication.

ACKNOWLEDGEMENTS

I would like to thank my Advisor Dr. R. P. Gardner for his guidance during the development of this work. I express many thanks also to Dr. John Mattingly whose conversations have always been insightful and demystifying on various nuclear related topics. Thanks go out to Dr. Jason Osborne for serving on my committee and teaching me more about statistics. I found your lectures more interesting and more rigorous then my undergraduate engineering statistics course. The skills I learn in your class with R and SAS helped me on my later courses. The entire Nuclear Engineering Department deserves many thanks for being so inviting and cozy these last couple of years to me. I would like to thank Gerald Wicks for providing and creating a multitude of sources that were needed for this work. I would also like to thank Scott Lassell for allowing me access to a Cobalt-60 source that I used as a reference source to determine the activity of the CEAR Co-60 source. I would also like to thank Dr. Dean Mitchell from Sandia National Laboratories who always expressed interest in my work. Thanks also go out to Dr. Andrey Berlizov who created MCNP-CP. There is no doubt that this exact work would have not come about had it not been for MCNP-CP. I express a lot of gratitude for my little sister Lindsay who mailed me all her nuclear engineering text books. I was able to go through graduate school without having to purchase any nuclear engineering text books. To all my colleagues here at the office, thank you. I want acknowledge all the help I received from Dr. Kyoung Lee. His help with the analysis and curve fitting made this work possible. I doubt I would have gotten this far this fast without his help and he has been a good friend. T. Wesley Holmes deserves credit for always wanting to help me out in the lab. It has been fun working with all the equipment in the lab. I would like to thank Mital Zalavadia for his help in setting up some of my experiments, a lot of those lead bricks were heavy to move around on my own. Personal thanks go out to my friends Cody and Johanna Peeples who welcomed me to North Carolina and provided so much hospitality while I found a home here. Cody you were the single most influential person in my decision to attend graduate school. Without your help on my graduate school application, I doubt I would have ever had the drive to finish it. I would have also never visited NC State had it not been for your wedding. To my mother and father I would like to express thanks for your support and words of encouragement throughout the years. Arminda De Hoyos thank you for your patience and belief in my abilities, usually more than even I feel certain about.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Objective

To develop an approach for applying the MCLLS approach to coincidence and sum pulse inverse spectral analysis using the DRF concept.

1.2 History

Previously work at CEAR (Center for Engineering Applications of Radioisotopes) has been done on the development of detector response functions (DRF).[3] These functions can save a tremendous amount of time because they are pre-calculated and the exact physics of the particle interactions happening inside the detector do not have to be simulated. It is usually sufficient for a simulation that uses DRFs to just simulate the arrival of the particles onto a detector. The appropriate DRFs are then applied for the given energies of the particles. The output of this function is then tallied.

1.3 Theory

Certain radioactive sources are said to have particle emissions in coincidence.[2] In reality these are particle emissions happening in so close a proximity of time that the detection system essentially detects a single event. In the case of gamma coincidence on a single sodium iodide detector, a sum pulse is created that represent the energy deposition of the multiple gammas entering. With regards to response functions, they are usually representative of a single energy.

In the case of Cobalt 60, its decay produces beta particles followed by the emission of various gammas. Some of the time these gammas are detected as a single sum pulses. If a simulation

is to produce spectrum that is comparable with that recorded in the laboratory via the use of detector response function, then it is believe that a single DRF created for the total summed energy to be inadequate. The supposition is that a convolution of the two single energy (1.1732 and 1.3325 MeV) detector response functions will create a new response function that will satisfactorily fit experimental data. This approach is perhaps more in line with the mathematics of convolving multiple Gaussian curves. This procedure usually yields a wider, more pronounced Gaussian curve even if it only involves multiple self convolutions. Because Cobalt 60 itself sometimes emits a single gamma ray that is the exact energy of the sum of the two individual common gammas, it is of interest to look at the differences between these two detector response functions.

1.4 Definitions

Decay Scheme shows the transitions in which a radioactive nucleus emits radiations to become less energetic and therefore reach stability. See Figure 1.1.

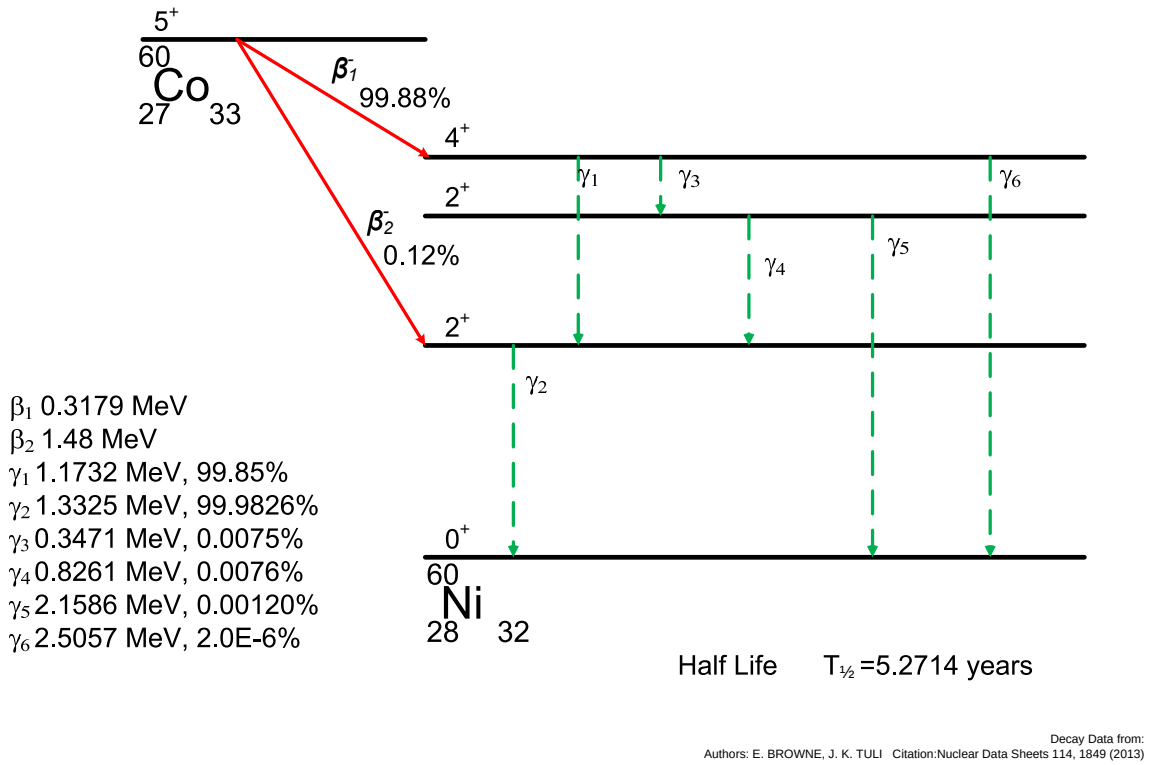

Figure 1.1: Co-60 Decay Scheme [7]

The **MCLLS Method** uses least-squares fitting from data libraries generated from Monte Carlo.[6]

Pulse Pile Up is a phenomenon in which a detector treats multiple pulses as a single composite pulse because the arrival time of the separate radiations is very close in proximity to each other.

G03 is specific code developed at CEAR used for the generation of Detector Response Functions. The code simulates the response of a detector that is based on a right circular cylinder with a source centered about the axial axis a certain distance away. G03 includes

PEAKSI is a specific code that uses CURMOD to obtain a Gaussian fit on experimental data. The Gaussian fit model can be composed of multiple Gaussian curves plus a constant, linear, or quadratic background. Parameters can be fixed or searched on with initial guesses. The code is useful for determining the full width half max in experimental data and centroids.

A web app version called WebPEAKSI of the code was developed as a proof of concept idea based on migrating legacy console i/o software to a web platform such that it can used inside a web browser.

CURMOD is a code developed at CEAR used to determine the parameters of a model or function with the minimum reduced chi-square value. It is based on work from P.R. Bevingtons book Data Reduction and Error Analysis for the Physical Sciences. It uses a Lavenberg-Marquardt algorithm to perform the analysis but unlike the algorithm detailed in the book, it also has the capability to search on non-linear parameters.

GShift Is a gain and zero shifting program for any channel-pulse height energy relationship to any other relationship including non-linear relationships.

CEARPPU A General Purpose Monte carlo code for modeling pulse pile-up distortion from high counting rates in nuclear instrumentation.

CURLLS is a code that uses library least-squares approach to determine the amounts of components in a photon spectrum. The program does this via the use of a subroutine model that calls on CURMOD. The outputs are calculated parameters, their standard deviations, their linear correlation coefficients and reduced chi-square value. The code can optionally implement a weighting scheme over ranges of the unknown spectrum.

CEARLLS is a code that used library least-square approach to determine the amounts of components in a photon spectrum. This code does not require CURMOD.

MCNP is a general purpose Monte Carlo code. It can simulate the physics of neutron, photons, electron transport.

True Coincidence involves multiple radiations from the same nuclear decay event. See Figure 1.2 where the color dots represent a decay event and the arrow represent radiations striking a detector.

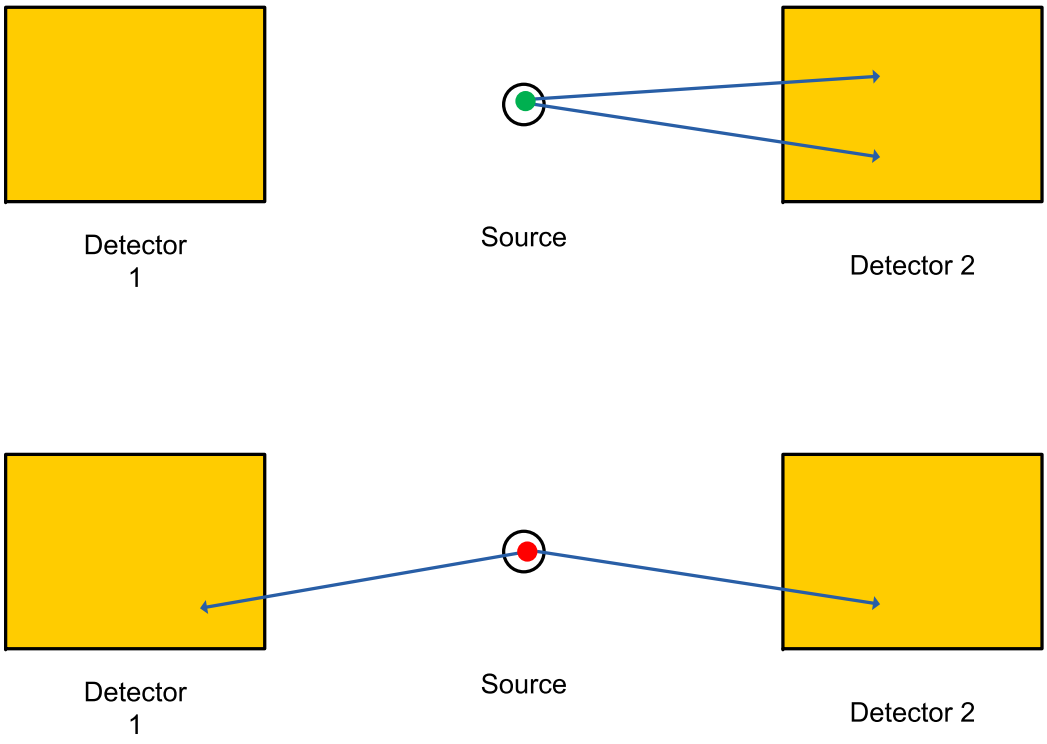

Figure 1.2: True Coincidence Scenarios

Chance Coincidence involves multiple radiations from two or more independent nuclear events. See See Figure 1.3 where the color dots represent decay events and the arrows emanating represent the associated radiation from such event.

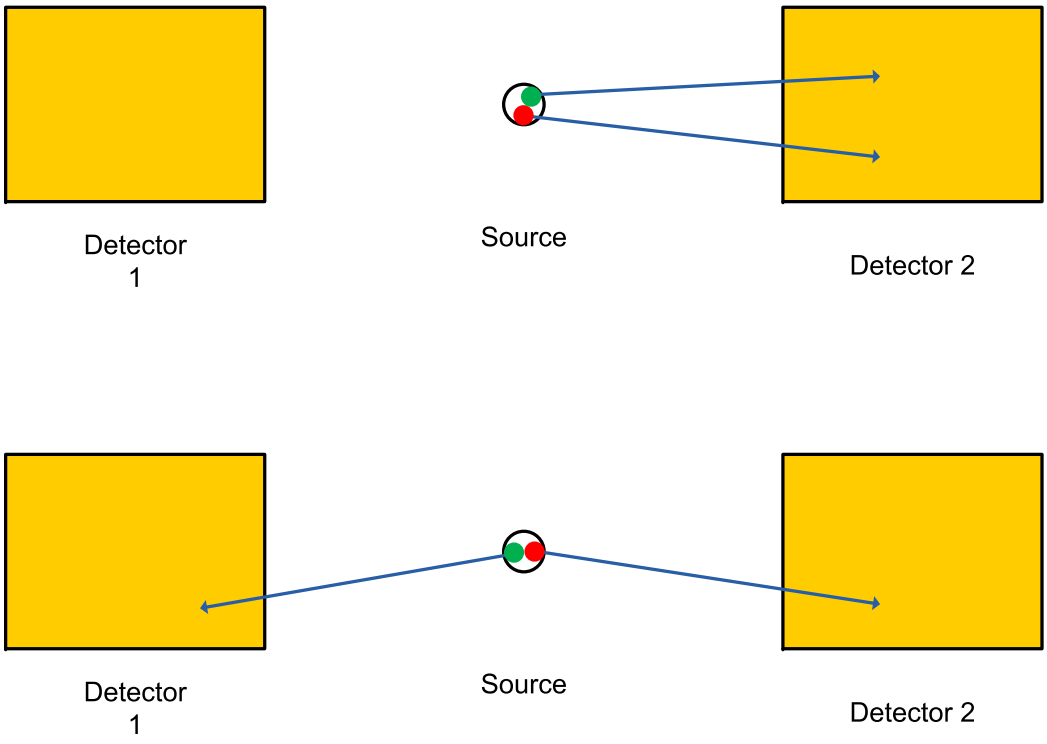

Figure 1.3: Chance Coincidence Scenarios

True Coincidence Summing involves summing of true coincidence events[4].

Detector - Detector Coincidence involves the tallying two incident radiations on separate detectors via the use of a gate trigger.

Detector Response Function (DRF) is a function whose output is the energy pulse-height distribution of a single energy incident radiation. The function itself is a probability density function.

Angular Correlation is defined as a correlation in angle of successive radiations in a cascade. Although the first radiations direction might be isotropic in the laboratory coordinates, the successive radiations are due to a cascade and their angle of emission will be correlated to the previous radiations angle of emission.

Sum Pulse is created by multiple radiations striking a detector around the same time. The detection system sees more energy deposition from the events but cannot distinguish them as

separate events and therefore produces a single summed pulse.

Gaussian Energy Broadening is a process applied to energy tally scores such that the energy score is reshaped. The shape is from the Gaussian distribution. This is done to more accurately simulate what a detection system ouputs.

MCNP-CP a software code created by Dr. Andrey N. Berlizov based on MCNP version 4c. It has added capabilities such as correlated particle sampling based on ENSDF (evaluated nuclear structure data file).[1] The radioactive source definition can simply be specified by typing its unique ZAM number. Other enhancements include the ability to form coincidence and anti coincidence tallies which can be based on cells that can have upper and lower level discriminators.

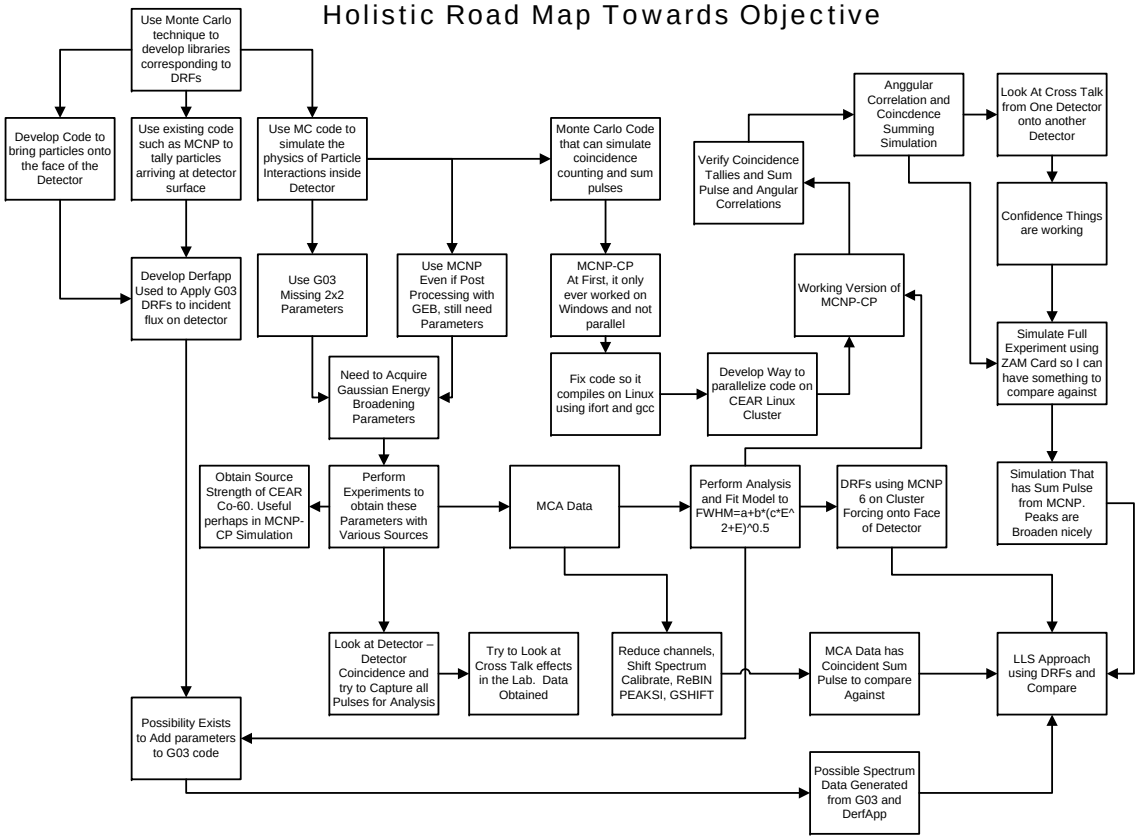

Figure 1.4: Roadmap towards goal

Chapter 2

Experimental Setup

2.1 Determination of Co-60 Activity

To determining the activity of the CEAR Co-60 source, a Co-60 source of known activity was used to obtain 4 MCA spectrums. These spectrums were produced by recording for 300 seconds the known source at distances of 30, 40, 50, and 60 cm. Background was recorded for 300 seconds as well. Later the same procedure was carried out for the CEAR Co-60 source of unknown activity.

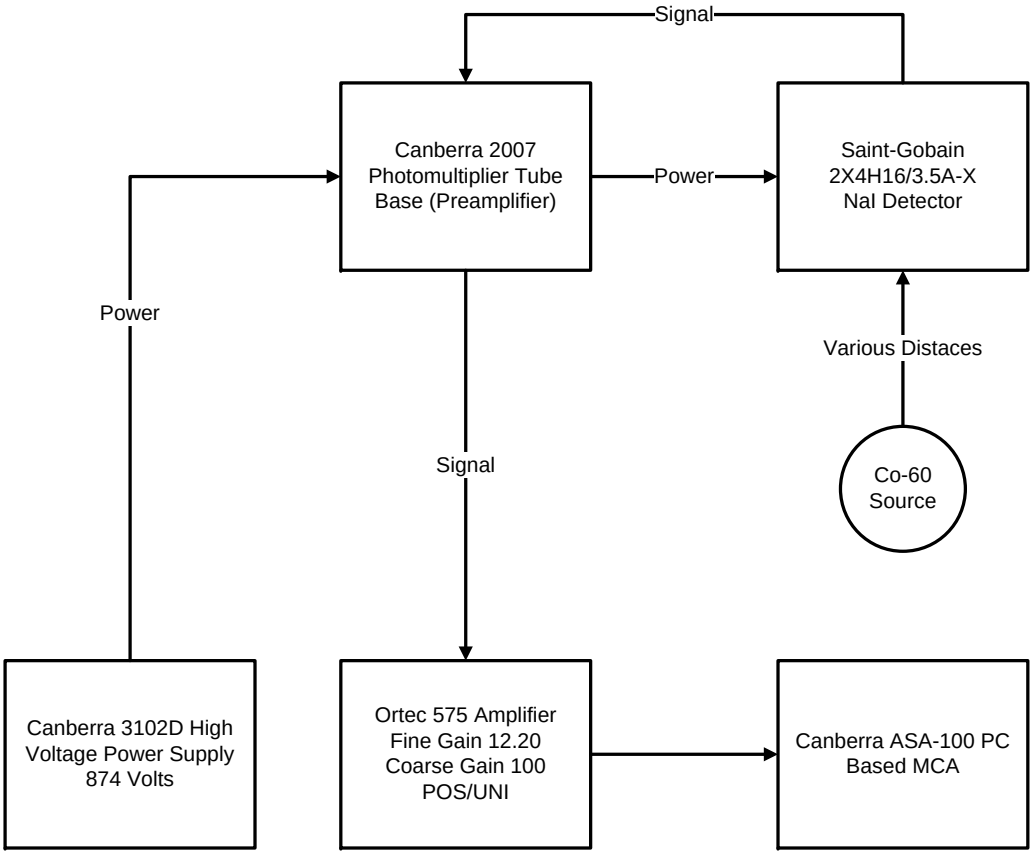

Figure 2.1: Setup used in experiment to determine Activity of Co-60

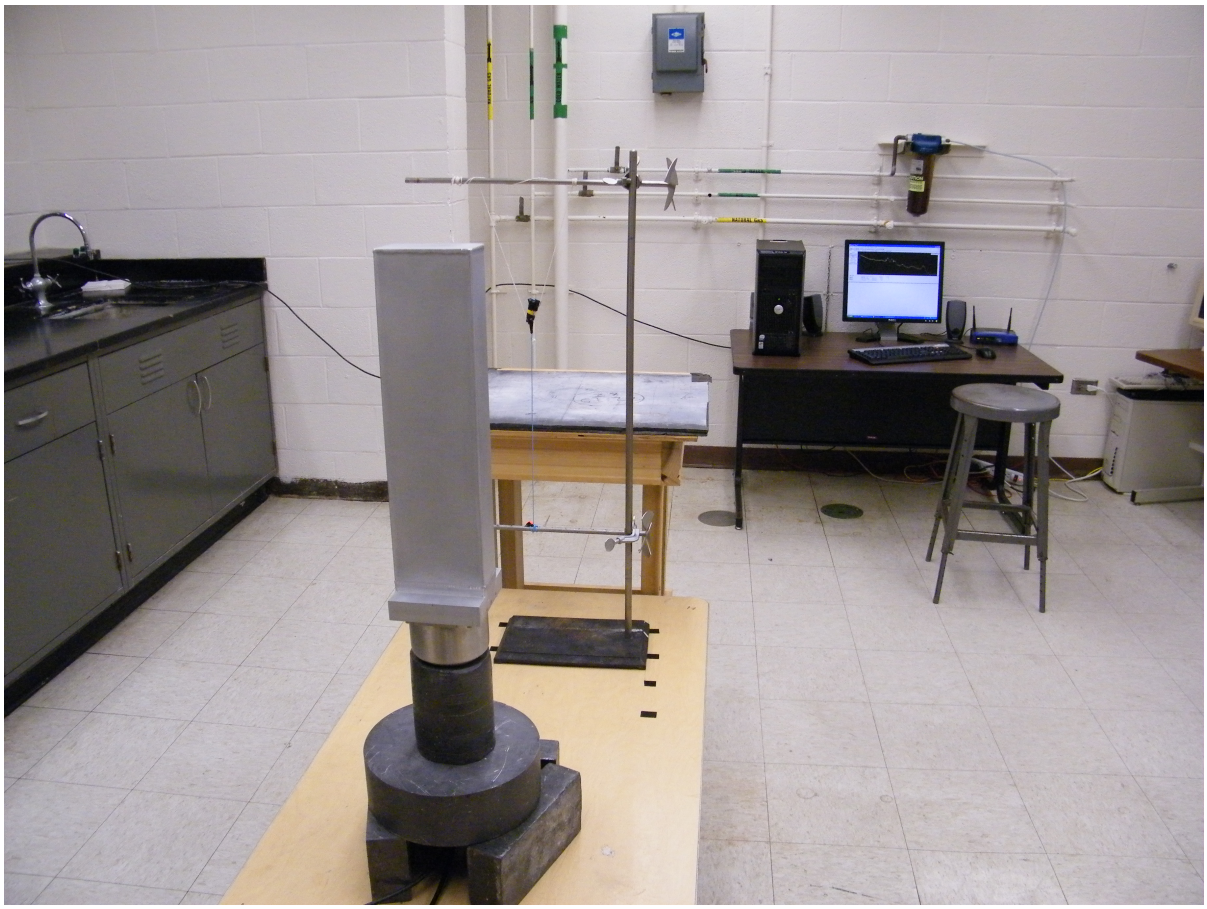

Figure 2.2: Picture of Detector used to determine Co-60 Activity

- The setting on the Canberra 3102D power supply was 870 Volts.
- The Ortec 575 Amplifier had a coarse gain setting of 100 and a fine gain setting of 12.20
- The Ortec 575 Amplifier was positioned for positive voltage and unipolar pulses.
- The known Co-60 source used has and activity of 0.9743 micro curies or 36.05 kilo Becquerels on the 15th of August of 2011.

These particular experiments were carried out on the 2nd and 3rd of July of the year 2013. A more thorough discussion on how the Activity of the CEAR Co-60 Source is obtained is given in the analysis section.

2.2 Calibration and Characterization

The idea was to record spectrum from various sources the yield peaks at different energies. This information is to be used to characterize how the detector behaves. Unlike the previous setup where there was a need for an Amplifier and a Pre-Amp tube base, the signal was taken straight out of the photomultiplier tube and onto the Pixie-500 DGF card.

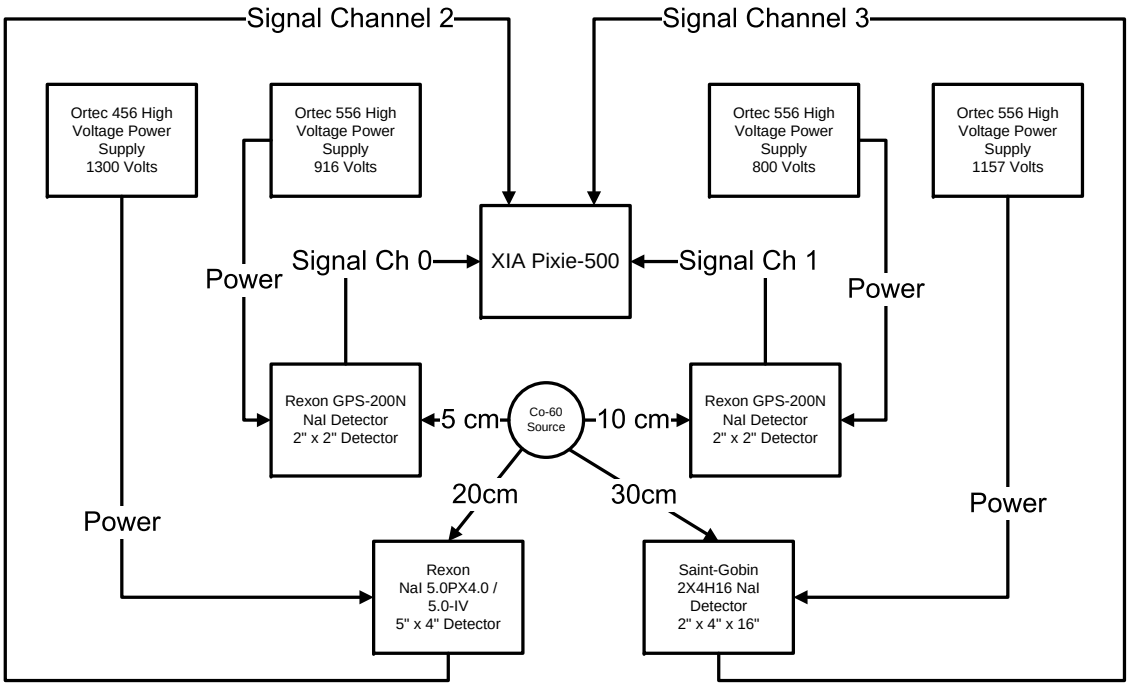

Figure 2.3: Setup used in experiment to collect data from 4 detectors

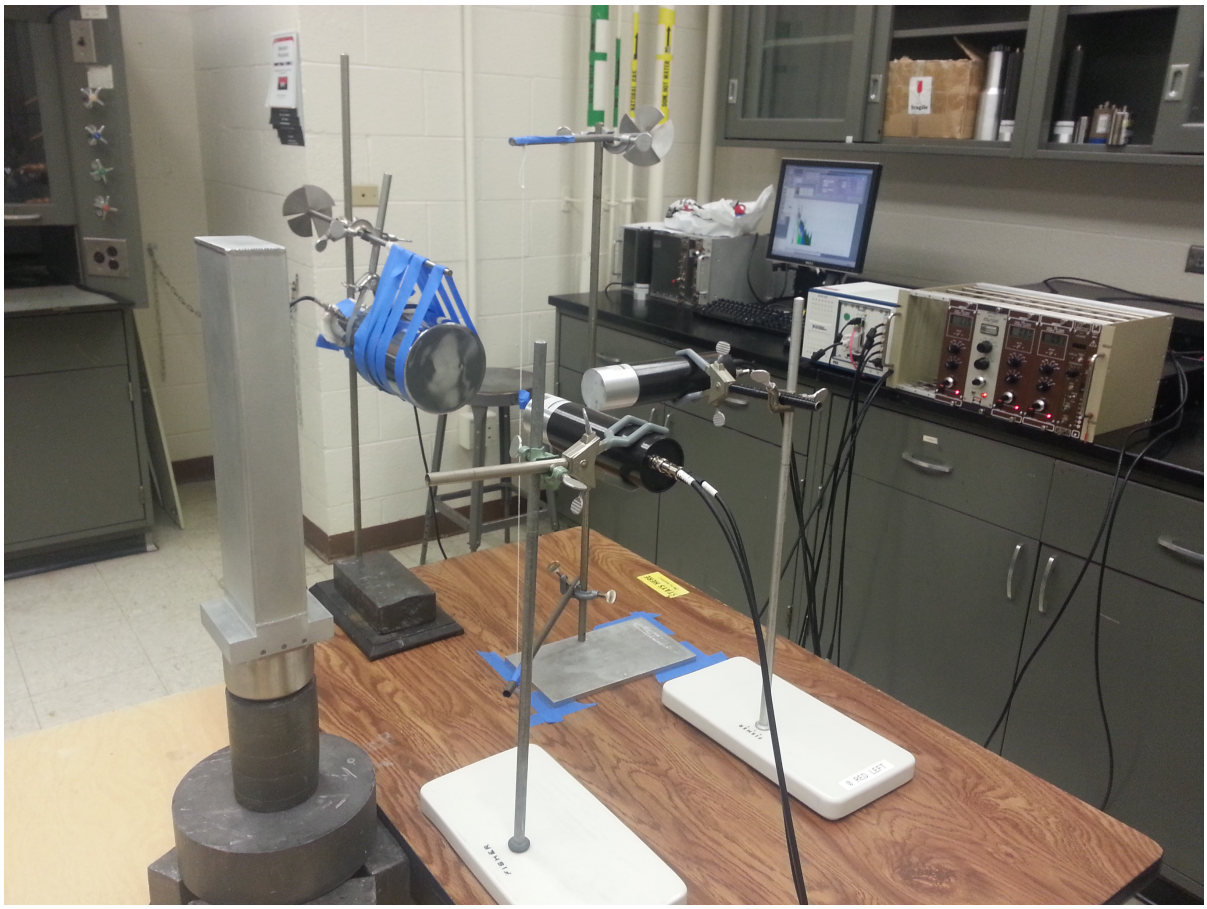

Figure 2.4: Picture showing collection of data from 4 detectors

The detectors are placed at 0, 90, 180 and 270 degrees from each other. The first detector was a Rexon 2 inch by 2 inch Sodium Iodide placed 5 cm away from a string that holds the source and connected to channel 0 on the XIA Pixie-500. A second 2 inch by 2 inch NaI detector was placed 10 centimeters away from the string. The third detector, a 5 inch by 4 inch Sodium Iodide, was placed 20 centimeters away from the sting. Finally a 2 by 4 by 16 inch rectangular box detector was placed 30 centimeters away.

The sources listed on Table 2.1 were used. A background measurement was taken without the source present before and after each source was placed on the string.

Table 2.1: Various sources whose spectrum was collected [7]

Isotop	Half-Life	Energy [MeV] First Gamma	Energy [MeV] Second Gamma
Co-60	5.27 years	1.1732	1.3325
Na-24	14.9 hours	1.3679	2.7535
Cs-137	30 Years	0.66162	
Au-198	2.70 days	0.41176	
Ba-133	10.51 years	0.356	Has various convolved peaks
S-37	5.05 min.	3.103	

2.3 List Mode and MCA data from the Pixie-500

Both of the Ortec 556 Power Supplies were set at around 1190 Volts in the positive bias position. The power cable was custom made having an MHV connector for the detector end and an SHV connector for the power supply end. The signal cables were made to be the exact same length using standard BNC connectors on both ends. On the end that attaches to the XIA Pixie-500, a silver coupler was uses to attach to the small cables that come with XIA Pixie-500. This is because the input connector to the XIA Pixie-500 is an SMA connector and not the traditional BNC found in nuclear instrumentation.

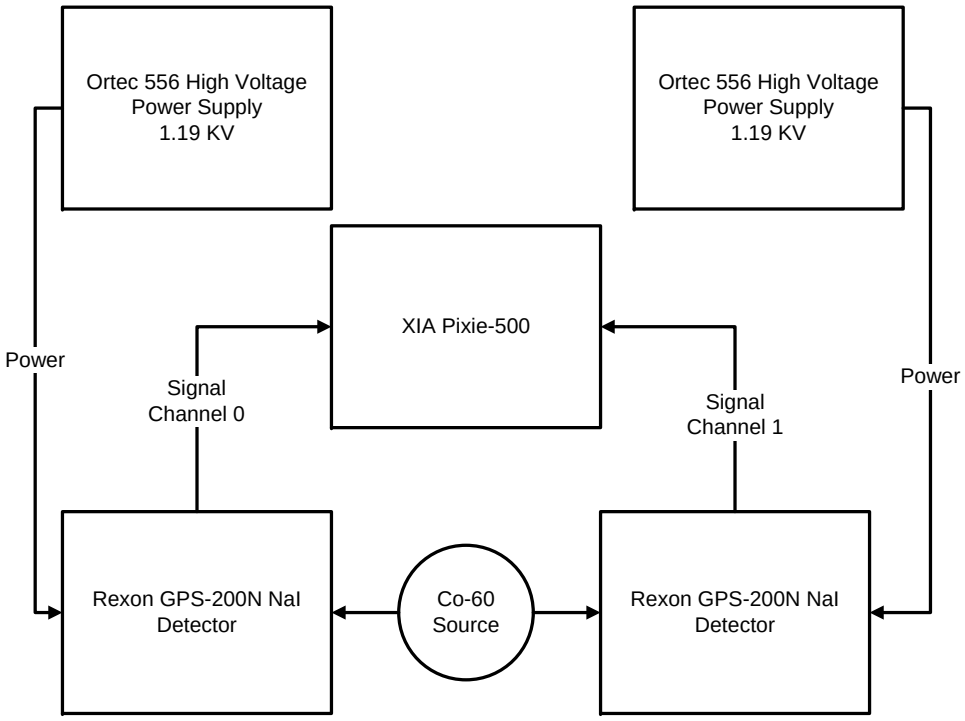

Figure 2.5: Setup used in experiments with 2 detectors

Hardware Settings on the Pixie-500 DGF PXI Card are as follows:

Channel 0:
JP101 - ("ATTN") this jumper block is set to short 1 and the middle position.
JP102 - this jumper block is shorted to select 50 ohm input impedance.

Channel 1:
JP201 - ("ATTN") this jumper block is set to short 1 and the middle position.
JP202 - this jumper block is shorted to select 50 ohm input impedance.

These jumpers were set this way because it significantly reduced the noise in the third floor lab. These settings made the tau values associated with the detectors significantly shorter as well.

Chapter 3

Simulations

3.1 Geomerty and Material Modeling

The specifications were figured out from e-mail correspondence with Rexon staff and the Data Sheet for the GPS-2000N Detector, also provided by Rexon. The aluminum thicknesses around the detector as well as the aluminum thickness on the face of the detector are both 0.0508 cm. The density of the aluminum modeled was 2.7 grams per cubic centimeter. The aluminum oxide powder reflector around the sodium iodine crystal is 0.254 cm thick. The aluminum oxide on the front face of the detector is 0.1016 cm think. The density assigned to the aluminum oxide powder was 3.97 grams per cubic centimeter. The BF-1000 rubber padding on the face of the detector is 0.1524 cm thick with a density of 0.1922 grams per cubic centimeter. The sodium iodide crystal itself is a cylinder with a radius of 2.54 cm, a length of 5.08 cm and a density of 3.667 grams per cubic centimeter.

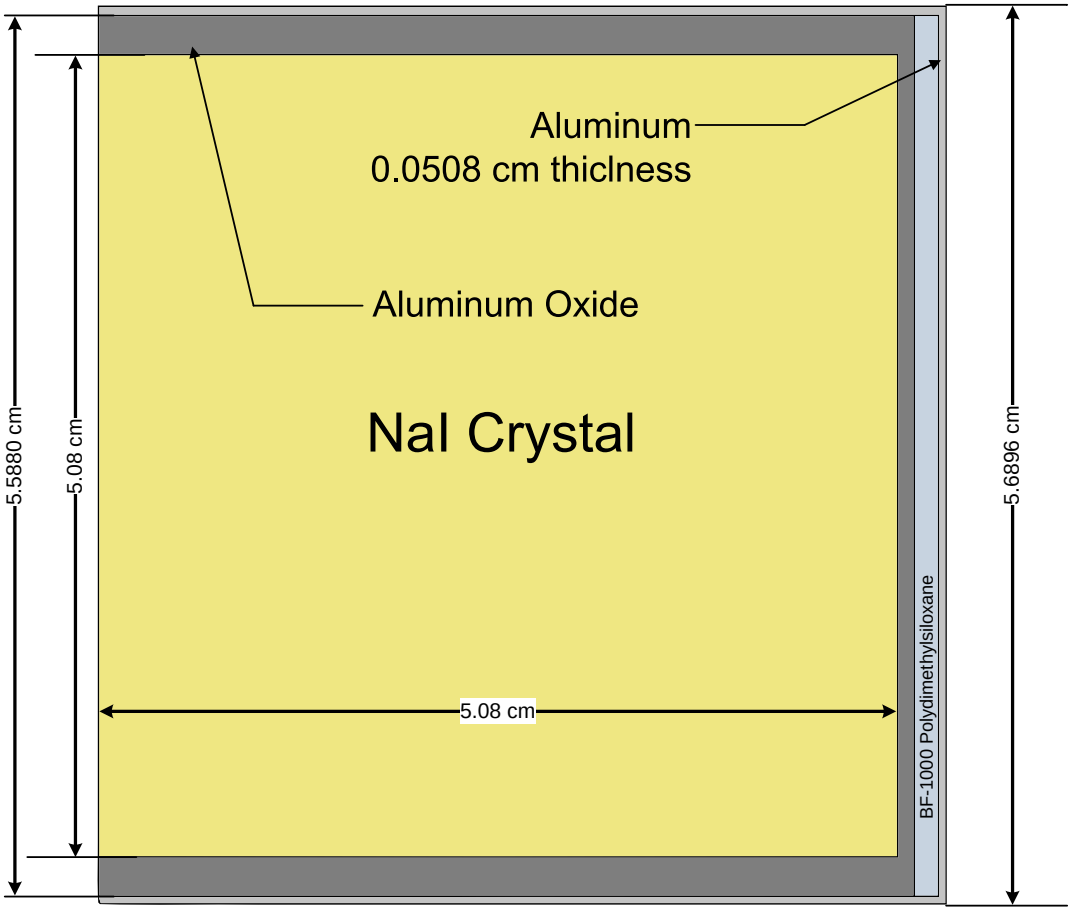

Figure 3.1: Inside view of geometry modeled in simulations

3.2 DRF Generation

3.2.1 G03

G03 was used to produce the detector response function for the model of a bare sodium iodide crystal. However the parameters belonged to a 3 by 3 inch detector. Hence this was only done for comparison purposes.

3.2.2 MCNP

MCNP 5 Version 1.60 was used to create the Detector Repose functions. These were created using the CEAR cluster and an MPI version of the MCNP executable. Direction forceing was implemented from a point source 5 centimeters away from the face of the detector. The distri-

bution was conical and onto the face of the detector.

3.3 Modeling Cross Talk between Detectors in Simulation

Looking at cross talk between detectors is of interest to some researchers. Various angles were looked at that included 67.5, 90, 112.5, 135, 157.5, and 180 degrees.

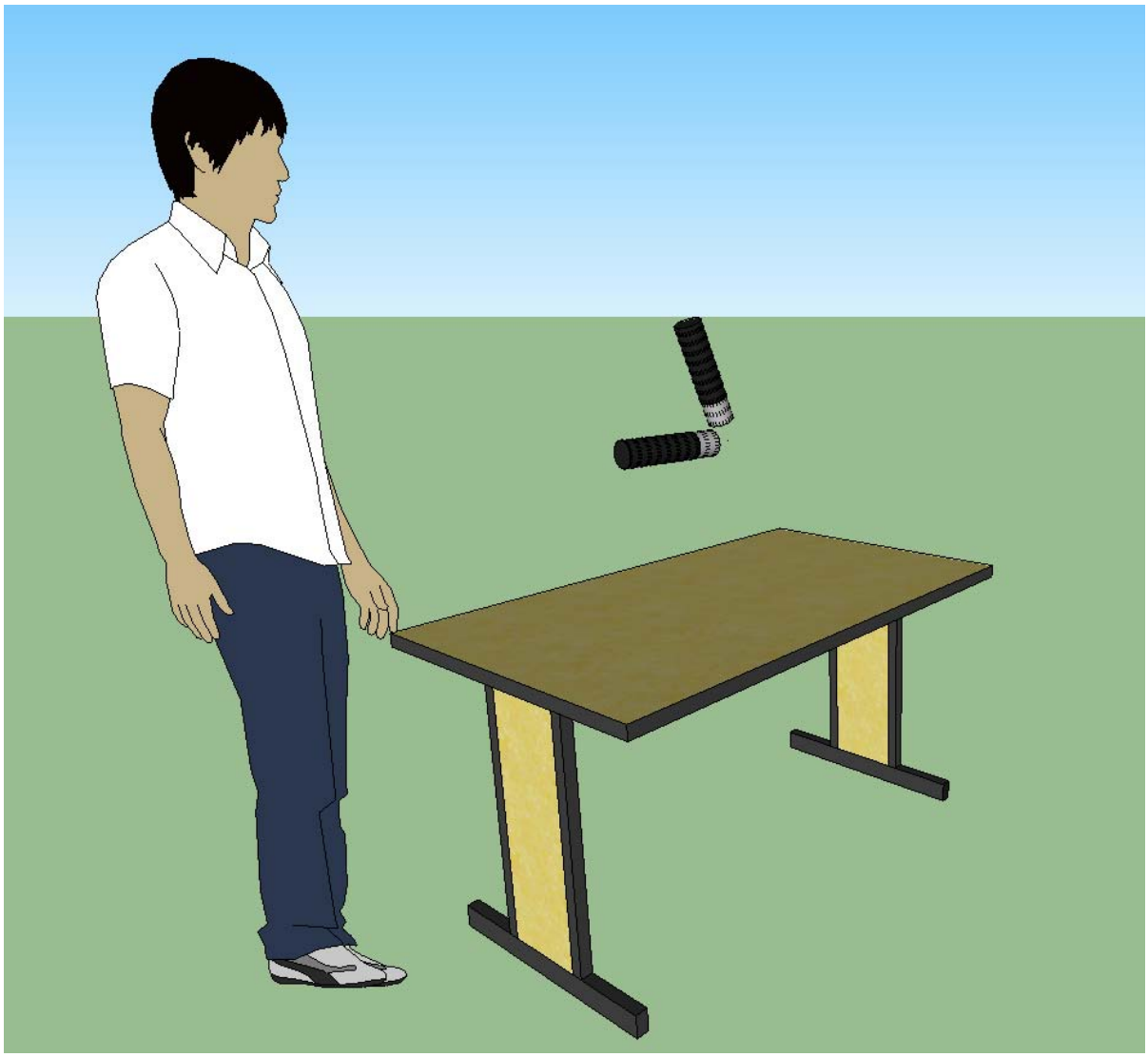

Figure 3.2: Illustration showing a horizontal fixed detector and a movable detector at 67.5 Degrees

To do this the particles were force exclusively to the face of a fixed horizontal detector from a point source 5 centimeters away. See Figure 3.2. A second detector was placed in the simulation that was also equidistant to the point source but would form an angle of 67.5, 90, 112.5, 135, or 180 degrees with the fixed detector. Therefore any particles depositing energy on the second detector was a result of scatter from the first detector.

3.4 MCNP-CP Angular Correlation between gammas

To perform an investigation on how good the angular correlation between gammas was, a ring of detectors was created for a simulation. See Figure 3.3. In this case, MCNP-CP was tested using features it has to tally coincidence between cell volumes. A total of 64 tallys were produced for each simulation run. A Total of 124 simulations of these types were performed using different initial random seeds. Each of the 124 output files had 8 tallies for 8 different angles, see Table 3.1.

Table 3.1: Various Angles that were tallied

CELLS	204	304	404	504	604	704	804	904	114	214	314	414	514	614	714
104	22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°							
204		22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°						
304			22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°					
404				22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°				
504					22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°			
604						22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°		
704							22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°	
804								22.5°	45°	67.5°	90°	112.5°	135°	157.5°	180°

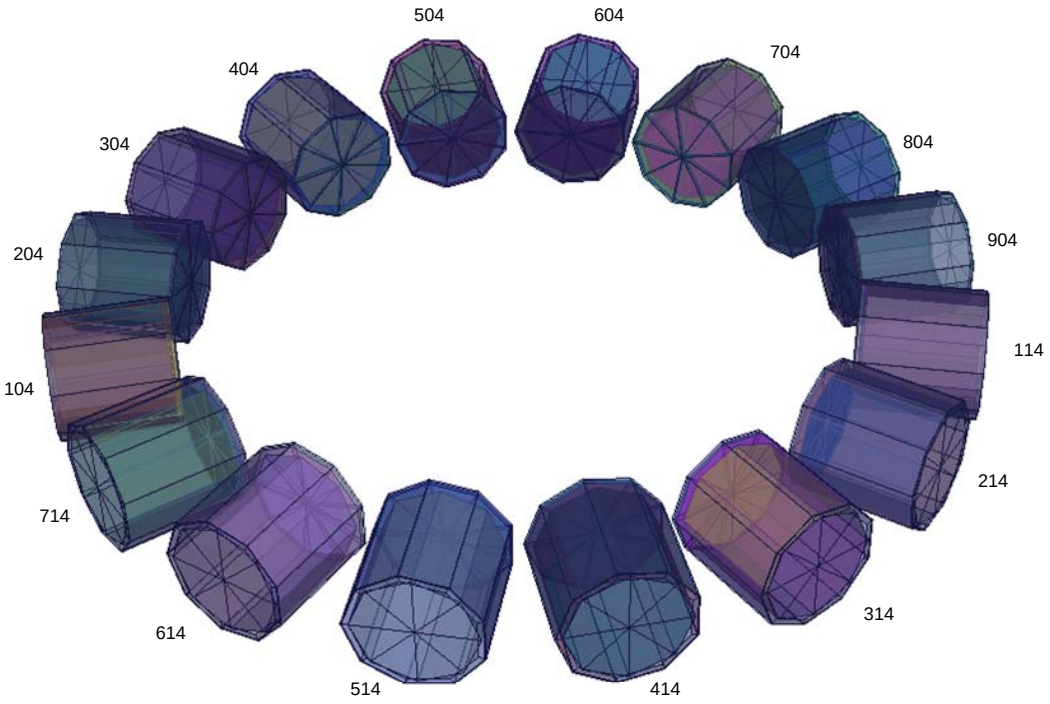

Figure 3.3: A Ring of Detectors around a point source modeled in simulation

Chapter 4

Analysis and Methods

4.1 Determination of Co-60 Activity

A spectrum that corresponds to 30, 40, 50, and 60 cm was adjusted by subtracting background for both the known and unknown activity sources. A region of interest that encompasses the two Cobalt 60 peaks was selected, in this case the region started at channel 353 and ended in channel 471. The total number of counts for these regions was obtained by simply summing over the number of channels with their counts. The ratio between the unknown and known was then used to solve for the activity of the CEAR Co-60 Source See equation 4.1.

$$A_{CEAR}\left(Co60\right) = \frac{TotalCounts_{ROI}(Unknown)}{TotalCounts_{ROI}(Known)} A_{Known}(Co60) \tag{4.1}$$

The activity of the known Co-60 source was corrected using equation 4.2.

$$A\left(t\right) = A_o \cdot e^{(-\lambda \cdot t)} \tag{4.2}$$

Where lambda was taken to be the natural log of 2 divided by 1925.20 days and the time t was taken as 688 days. This was the difference between August 15th 2011 (the day the activity of the known source was recorded) and July 3rd 2013 (the day this particular calculation was done). The average of the calculation was then taken between the 30, 40, 50 and 60 cm cases.

4.2 FWHM model and the A B C's

To create parameters for the following equation, FWHM= $a + b\sqrt{E + cE^2}$.[8]
The data from the experiment was used in the following manner. First a net count for the

particular isotope was determined by subtracting a corresponding back ground. Next, the net spectrum was energy calibrated to the peaks of the known energies for the particular isotope. This was done by performing a Gaussian fit on the curves and taking the centroids as the place where the full energy deposition occurs. For example in the case of Cobalt 60, the energies looked at were 1173 keV and 1332 keV. This was done for Sodium 24, Cesium 137, Sulfur 37, and Barium 133. Thirdly the full width at half maximum was recorded for each distinct energy. With Multiple data points, a best fit to the model equation was performed. This was done multiple times with different software to arrive at better initial guesses until finally the resulting parameters were able to fit the data well.

4.3 Simulation

Comparison between the simulation data and experimental data must be done on an energy scale. Rebinning of experimental data was done. The experimental data was chosen to rebin rather than the simulation data because because the experimental data posses higher fidelity than the simulated data. Going the other way around did not make sense. The experimental data was made to fit 1024 channels ranging from 0 to 3 MeV for the case of Co-60. The parameters obtained for the FWHM model in the experimental analysis were used in the simulation via the GEB card. Output of the simulation contained both Gaussian energy broaded data and non broaded data. The approach to broaded data in post processing was not employed although a comparison between the two techniques might prove interesting if discrepancies arise.

4.3.1 MC Simulation Generated DRFs

The detector response functions that were finally employed were the ones generated with MCNP 5 Version 1.60. These were generated for the model that included the aluminum can, the reflector, rubber padding and Sodium Iodide Crystal. Neither photomultiplier tube nor its casing were modeled. To reduce the time it took to produce these DRFs, the simulated source for the corresponding single energy was forced onto a conical distribution arriving at the face of the detector. Also these simulations were carried out using MPI on 155 processors on the CEAR Cluster. Ten billion source particles were generated for each of the single gamma-ray energies associated with Cobalt 60.

To form the DRF that corresponds to the sum pulse in Cobalt 60, the two DRFs corresponding to the energies of 1173 keV and 1332 keV were combined. Several methods were thought out, amongst using the rejection method to sample from the two, adding upward sloping diagonals on a matrix formed from a type of matrix multiplication operation, and using wave analysis

software to convolve. However the final convolution used to generate the DRF corresponding to a summed energy of 2505 keV was done rather easily with a nested loop. See appendix under custom code for convo.c. The reasoning behind using this method was that all the data points were positive and the DRFs for the 1173 and 1332 keV gamma-rays were themselves tabular data. The detector itself sees the incoming gamma-rays as a manifestation from independent events. Because of this they were treated as probability mass functions that could be convolved using the following formula:

$$P(n) = \sum_{k=0}^{n} p_x(k) \cdot p_y(n-k) \tag{4.3}$$

where p_x and p_y are the probabilty mass functions to be convolved and P is the result.

Chapter 7 of reference [5] has a very thorough discussion on convolving probabilty mass functions.

4.3.2 Running MCNP-CP

In order to run MCNP-CP efficiently the CEAR cluster was used in a pseudo parallel manner. A BASH script was used to run multiple instances of a particular simulation starting with a different set of random numbers on multiple computers. A second script was then used to collect the data. Finally custom written program developed in C were used to average the results and propagate the error. This type of procedure was carried out twice. Once for the simulation of a ring of 16 detectors around a point source, and a second time to produce the data for decay of Cobalt 60 at 5 centimeters away from the detector. The later of these simulation yield a spectrum with a sum pulse corresponding to the addition of the two prominent gamma-ray energies from Cobalt 60. The scripts and programs used are provided in the appendix.

Chapter 5

Results

5.1 CEAR Co-60 Activity

The activity of the source was determined to be 130 kBq with a sigma of 1.8 kBq or about 3.51 micro curies on July 3rd 2013. The deviation is probably not very meaningful without having known the known source's error to carry out error propagation properly. However this estimate was good enough.

5.2 MCNP vs. MCNP-CP

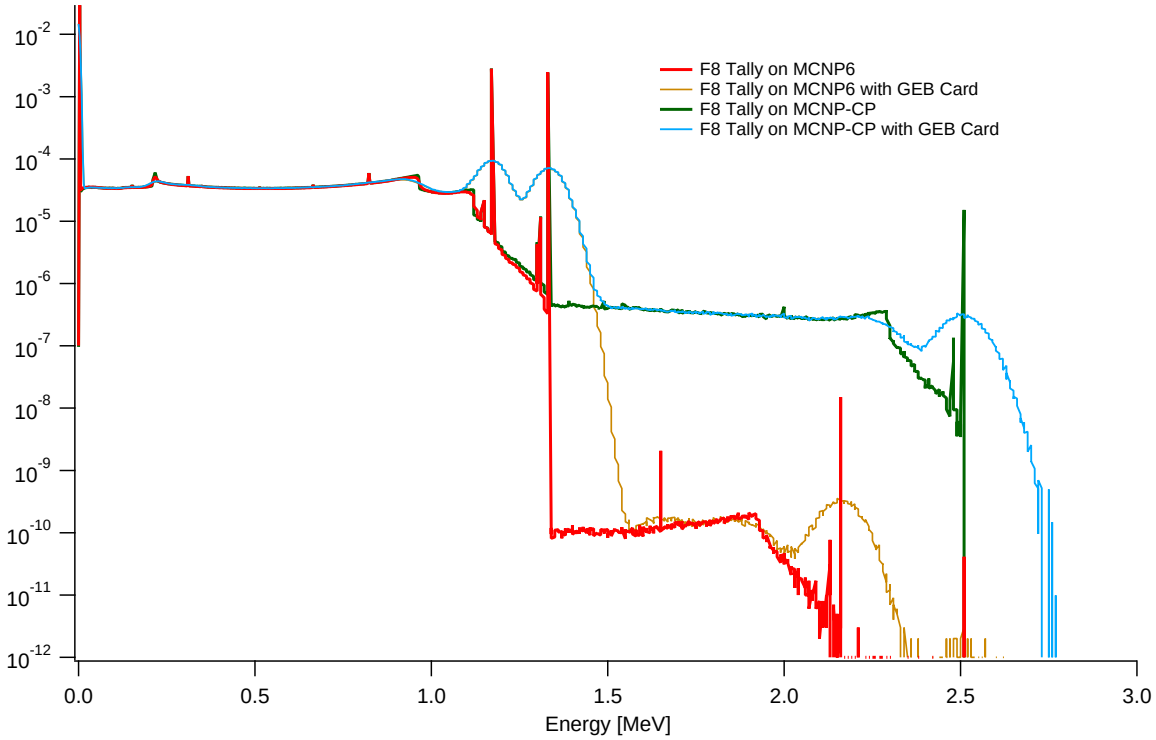

Figure 5.1: MCNP vs MCNP-CP with and without GEB

Figure 5.1 shows a comparison of MCNP and MCNP-CP. The MCNP simulation shows the other energies listed in the decay scheme, Figure 1.1. Notice that these energies above 1.3 MeV are the low emission yield. The output from MCNP-CP does not appear to show these energies but does show a sum pulse corresponding to 2.5 MeV. This is closer to what is observed in the laboratory with experimental data.

5.3 Cross Talk Simulation at Various Angles

The plot in Figure 5.2 compares spectrum for the case where all particles are forced onto one detector at zero degrees and 5 centimeters away from the source. This was done via a conical distribution on the face of the detector. A second detector is rotated at various angles and the energy deposited on this detector is purely from scatter off of the first detector.

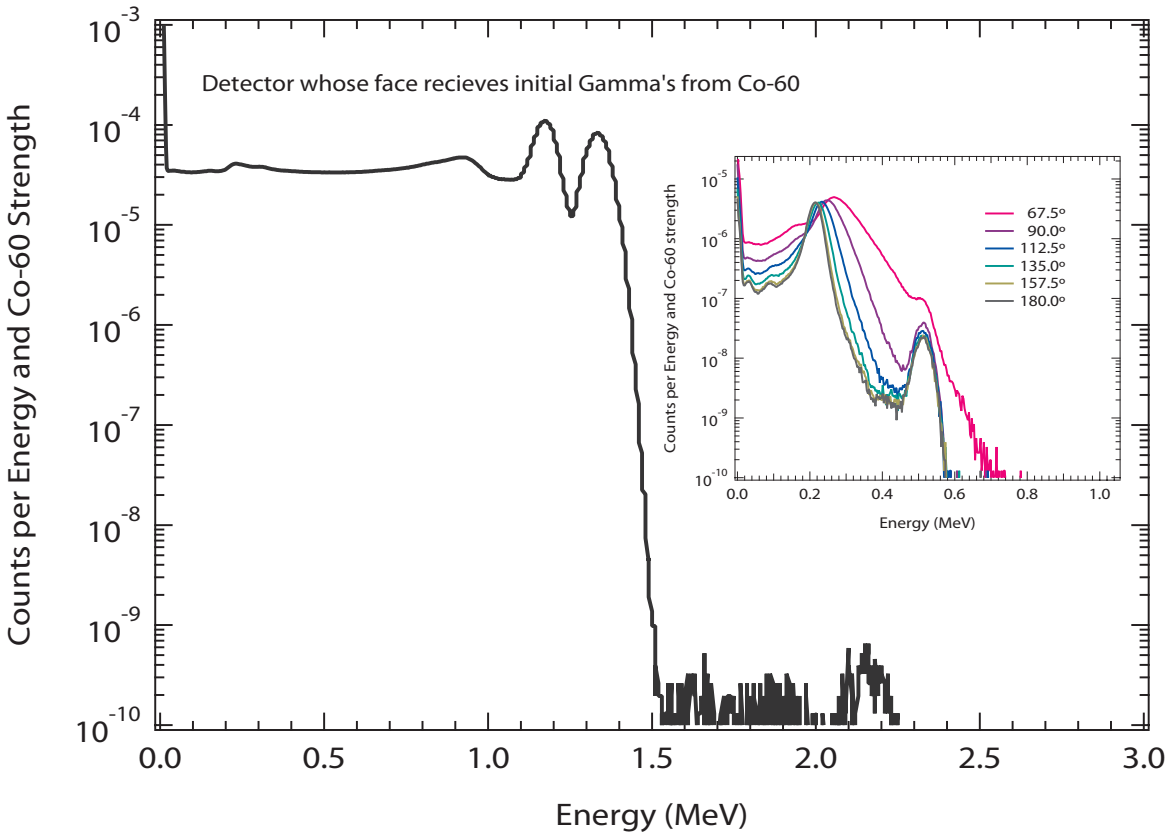

Figure 5.2: Cross Talk introduced into detector from scatter off of first detector

5.4 Angular Correlation from MCNP-CP Simulation

In Figure 5.3 the theoretical Model is the blue line. The red dots represent total count ratio between the angle of interest and the 90 degree case for various angles. See Figure 5.3. This calculation is not possible within reasonable time using a single instance of MCNP-CP on a single computer. The entire CEAR Cluster was used with the help of BASH scripts and a separate C code to run multiple instances of MCNP-CP. This data produced 124 output files with 64 tallies each. This amounted to around 75 Gigabytes of data. Transferring the data itself after the simulations were ran took around an hour and a half. The entire run took 3 days using 31 nodes on the cluster.

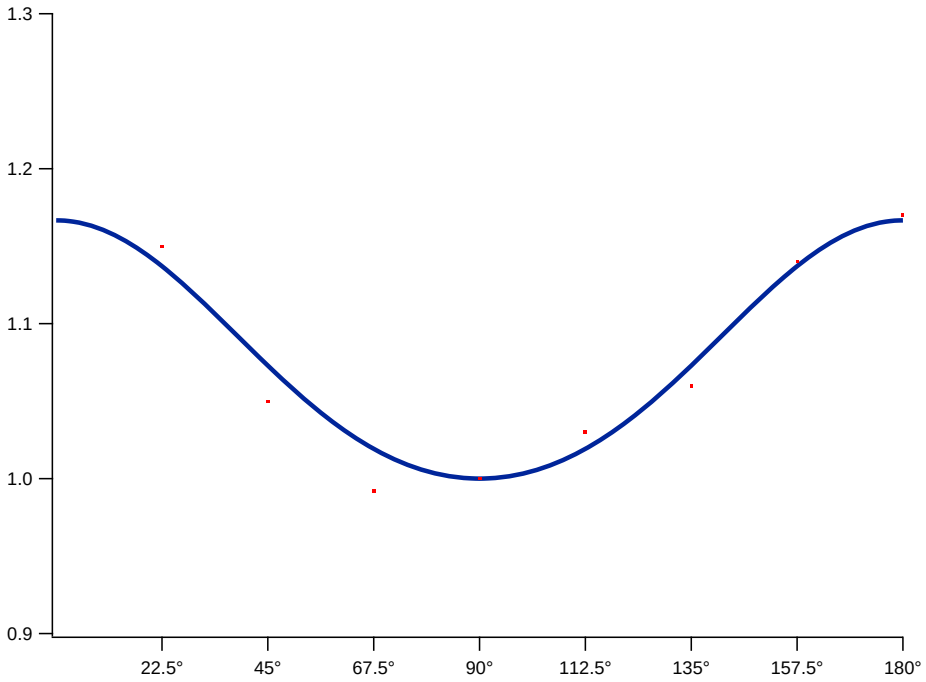

Figure 5.3: Angular Correlation Plot of Theoretical Model and Simulation Data

5.5 DRFs

5.5.1 G03 Generated DRFs

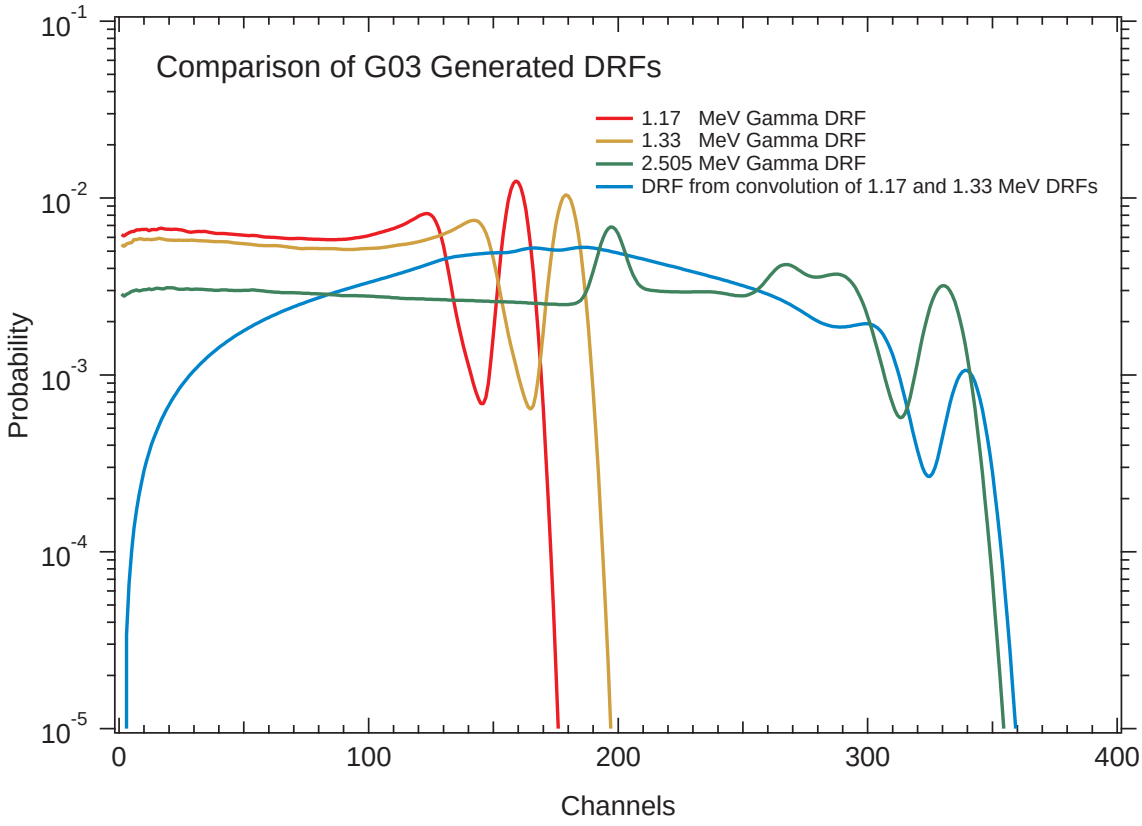

Figure 5.4: DRFs Generated with G03

5.5.2 MCNP Generated DRFs

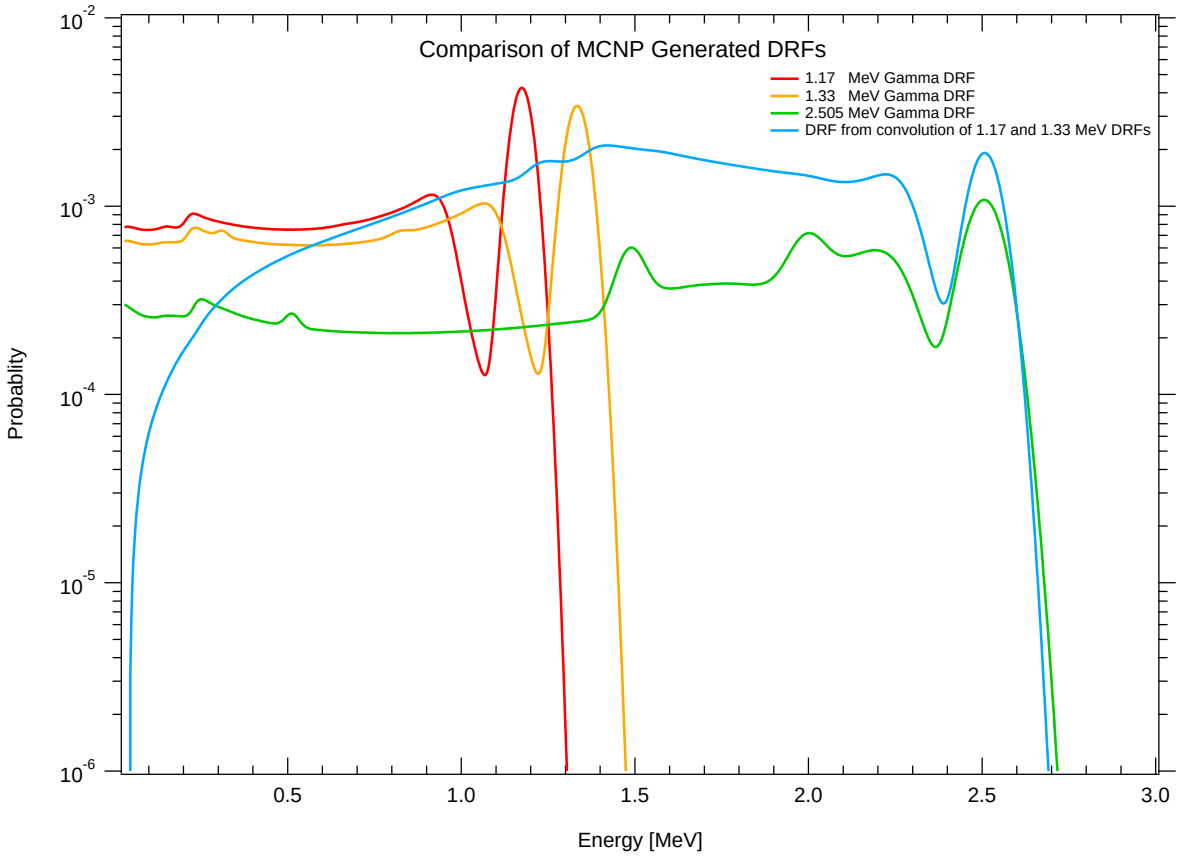

Figure 5.5: DRFs Generated with MCNP

5.6 Final Result

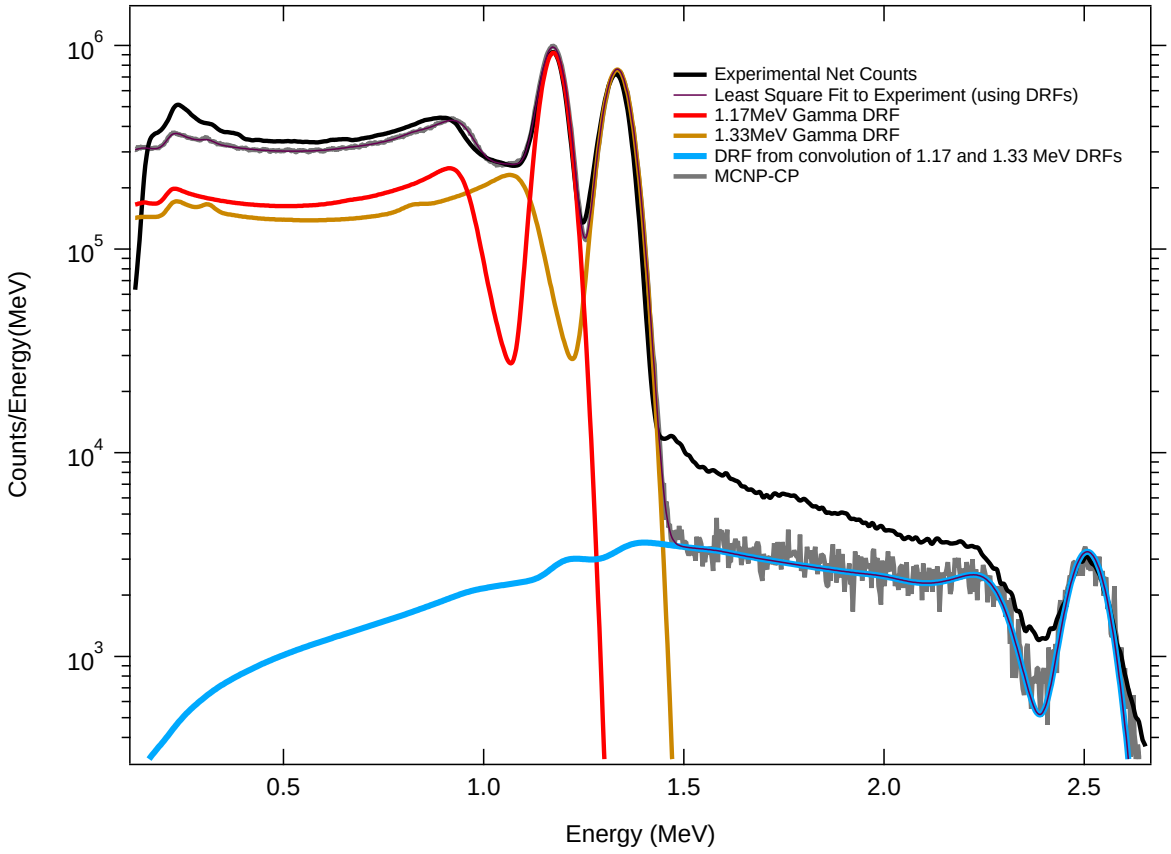

Figure 5.6: DRFs fitted to experimental spectra with MCNP-CP spectra super imposed on top

The final result Figure 5.6 shows various fits to the experimental data (black solid line). The individual DRFs as well as the least-squares fit to the experimental data is shown. The output of MCNP-CP was the final layer to be added to the graph. The interesting point is that the least squares fit was only done to experimental data and the MCNP-CP data seems to match this fit.

Chapter 6

Discussion

A useful application of DRFs might arise when subtracting from experimental spectrum. If what appears to be a sum pulse is present it might be possible to determine if it is cause by a summation effect from single energies due to the difference in the full width at hald maximum.

Pulse extraction code for the binary output file from XIA Pixie-500 was written in C and included in the appendix. MCNP-CP also has a list mode feature but a comparison was not done.

REFERENCES

[1] Andrey N Berlizov. Mcnp-cp: A correlated particle radiation source extension of a general purpose monte carlo n-particle transport code. In *ACS symposium series*, volume 945, pages 183–194. Oxford University Press, 2007.

[2] Robley Dunglison Evans and Atome Noyau. *The atomic nucleus*, volume 582. McGraw-Hill New York, 1955.

[3] Robin P Gardner and Avneet Sood. A monte carlo simulation approach for generating nai detector response functions (drfs) that accounts for non-linearity and variable flat continua. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 213:87–99, 2004.

[4] Gordon Gilmore. *Practical gamma-ray spectroscopy*. John Wiley & Sons, 2011.

[5] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1998.

[6] Weijun Guo. Improving the mclls method applied to the in vivo xrf measurement of lead in bone by using the differential operator approach (mcdolls) and x-ray coincidence spectroscopy. 2003.

[7] Brookhaven National Laboratory. Nudat 2.6 database. http://www.nndc.bnl.gov/nudat2/. Accessed: Jan. 2013.

[8] LANL X-5 Monte Carlo Team. *MCNP-A General Monte Carlo N-Particle Transport Code. Version 5*. Los Alamos National Laboratory, Los Alamos, NM, USA, 2003. LA-CP-03-0245.

APPENDICES

Appendix A

About the CEAR Cluster

A.1 History

This is the third cluster CEAR has had. It is based off of some improvements on the second cluster which was donated to CEAR. The previous cluster was more than double the size and considerably loader. One often had to wear ear protection when entering the room where it was housed. This new cluster was rebuilt in the fall of 2009 but has since gone through several upgrades. When building the new cluster care was taken to distribute the load onto multiple electrical circuit breakers. The previous cluster had significant power issues because of the size, power consumption, and electrical power distribution. The new cluster sports a shared file system that is available to all the nodes as well as gigabit Ethernet interfaces. Its primary purpose has been to run and develop Monte Carlo codes which are usually inherently parallelizable.

A.2 Hardware

The CEAR cluster consists of 41 nodes (node100 through node140). Each node is essentially the same having an AMD Phenom 9950 Quad Core 2.6 GHz CPU and 4 Gigabytes of RAM. The decision was made to use consumer level hardware and inexpensive motherboards which featured onboard gigabit LAN Ethernet interfaces as well as onboard video display ports. Each node also has its own hard drive which is partitioned in 3. The first partition is swap space and is around 4 gigabytes in size. The second partition houses the primary local file system for the node. This partition also has the nuclear cross sections installed for faster read access to the system. The third partition is mounted as /local and is space reserved for users and applications that require local disk access. The directory /home is a mount point for the shared network file system housed on a separate machine whose hostname is nfsserver. The NFSSERVER houses

the users file system on multiple hard drives which have partitions in various RAID setups. See the Section How the File System Works on the CEAR Cluster under the Appendix Paralleling Code Manually on the CEAR Cluster for a better explanation.

Teletraan1 is the most recent computing host developed with the idea of a possible migration path towards new hardware. It features two CPUs each with 16 Cores for a total of 32 Cores. Its memory is at 256 Gigabytes of RAM as of now. The host also has a high speed solid state drive. The home folder is again mounted from the NFSSERVER.

Logger is a simple machine with one purpose, to log information from all the other nodes and computers. This is done via the SNMP service that runs on all the hosts. When problems arise on the network, the log files kept here are a good starting point. Things of interests that are recorded here are CPU and memory Usage as well as any hardware failures reported by any other host. Logger has the ability to send SMS and e-mail messages to report issues.

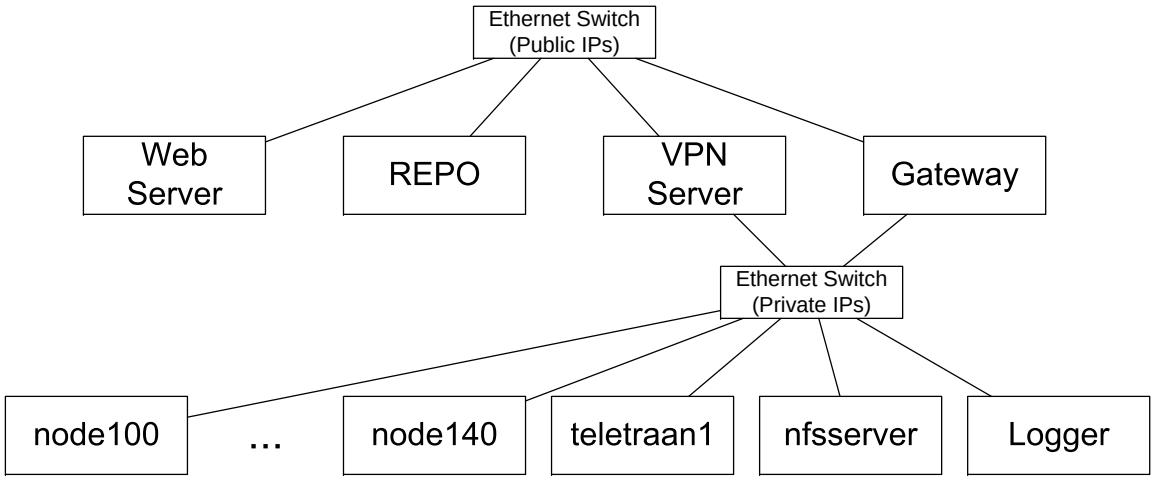

Figure A.1: Diagram showing Layout of CEAR Cluster

Both the Gateway and VPN servers provided internet connectivity to the cluster. This is useful as this cluster is on a private network not accessible publicly. Users of the cluster are provided with certificates to access the CEAR cluster network. These certificates are either installed on a router that has a VPN client or the users computer along with a VPN client. Connectivity amongst all the hosts is handled by a gigabit managed switch that has both a

terminal interface via a serial port and a web interface. The switch itself also runs an SNMP service and reports to LOGGER.

A.3 Software

With the exception of the VPN Server and Gateway, all the machines run some version of Slackware Linux. There have been several upgrades performed and most hosts run on Slackware 14. Most of the machines use a 32-Bit version of the OS with the exception of Teletraan1 which itself runs a 64-Bit version of the operating system. This decision is due in part because some work has to be done on custom codes for them to run on 64-Bit versions of the OS. Teletraan1 has to run on a 64-Bit OS because of the amount of memory it has. Teletraan1 is also a good place to test migration of the code to work on 64-Bit systems. The VPN Server and the Gateway run pfSense, a distribution of FreeBSD used for routing purposes. Manual DHCP (Static DHCP) is enabled so the nodes can get the same IP address based off of their MAC address. However the machines are also usually set with a static IP address. These machine also allow for network booting. They send out information to allow PXE booting from a folder shared on the NFSSERVER. This is done for repairs and diagnostics on the nodes. Both VPN Server and Gateway also run an NTP Server used to keep the time on all the computers on the CEAR Cluster network synchronized. The MPI libraries used for distributed computing come from MPICH version 2. They were built from source code using the GNU C compiler and the Intel Fortran compiler. A Host named REPO is used as a repository with software version control systems for the management of custom written code. It features all the standard packages such as git, cvs, and subversion. The Web Server is used to host the site www.cearonline.com. This is done with the Apache web server, php, and MySQL. Drupal is used as both a content management system and code frame work for custom written php modules. Every released subversion of MCNP5 is compiled for use on the cluster in either single or distributed mode which includes both OpenMP and MPI. The latest version of MCNP 5 and 6, MCNPX and MCNP4C are also installed.

A.4 Management

Management is made easier via the use of custom written BASH scripts and CRON jobs used to do administrative things such as adding users, backing up files, and synchronize between various nodes. Some of these scripts are used to setup SSH keys for newly added users as well as creating their locally addressable space on each node. The administrative versions are kept in /usr/local/sbin while utitilies that can be ran by normal users are kept in /usr/local/bin

A.5 Maintenance

There has been a lot of maintenance work performed over the last couple of years on the CEAR Cluster. The biggest culprit of headaches has been inflated capacitors. These has usually been replaced by desoldering and resoldering new ones on motherboards. Lately because of time constraints is has been easier to replace entire motherboards which have been running around forty dollars or so. Because of the constant use at near 100 percent of CPU usage, the capacitors seem to last about a year and half to two years before needing replacing. Recently better cooling has helped decrease some failures.

Only one CPU on a node has ever needed replacing and it was under warranty via AMD. A total of six power supplies have failed over the years. Opening up the power supplies revealed that problems were more than likely also caused by faulty capacitors. The decision to just replace the entire power supply unit is usually taken. Hard drives have also experience failure. No data has been lost thus fare and down time has been minimum due partly in fact to the used of RAID 5. Failed hard drives are tested via the use of low level diagnostics software. When possible they are secure erased and/or zero wiped. If not possible they are opened and made unusable and their magnets are removed.

When PXE booting from the network on a NODE it is possible to run various diagnostics utilities. These utilities can test RAM and provide one with the SMART tables from the hard drives. Careful attention is paid to look at the grown defects list and SMART tables for possible pending sector relocations. If any are found the drive is secure erased and then zero wiped and removed from the system. When a machine is experiencing faults its usually taken off of the rack and inspected visually also for capacitor problems and full diagnostics are done. When removing a system from the cluster, its entry is usually removed from the hydrahosts file so that it does not get used by MPI enabled software. Once repaired or replaced the system is network booted once more. From here it also possible to restore a generic system onto the hard drive of the computer. When rebooted system particularities such as unique IDs and keys are restored for the host before reintegration onto the cluster. The utility fsarchiver is used as it can format and restore a file system simultaneously.

A.6 Future Improvements

The use of uninterruptible power supplies has been considered; however when pricing them out it seems more lucrative to invest in more computing hard ware. Perhaps it is time to perform an analysis and reconsider purchasing some. Surge protectors have served well but few people know

how to restart cluster jobs from run tapes. When all the codes are tested and shown to work well in 64-Bit computing systems, it will be wise to try to migrate to a pure 64-Bit architecture. Other MPI enhancements are possible but require thorough testing. Adding OpenMP and MPI capabilities to custom codes would also benefit all who continue to use the cluster.

Appendix B

Custom Codes

B.1 reduce.c

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i,n,x,y,z,u,v,w;
  int *array=NULL;
  FILE *in, *out;


  if (argc !=3 )
  {
    fprintf (stderr, "Correct ussage is:\n");
    fprintf (stderr, "%s inputfile outputfile \n",argv[0]);
    exit (1);
  }

  if ( (in=fopen(argv[1], "r")) == NULL)
  {
    fprintf (stderr, "Can't read %s.\n", argv[1]);
    exit (1);
  }

  if ( (out = fopen (argv[2], "w")) == NULL )
  {
    fprintf (stderr, "Can't write %s.\n", argv[2]);
    exit (1);
  }

  i=0;
  while (!feof(in))
  {
    if (fscanf(in, " %d %d %d\n", &x, &y, &z) != 3) break;
```

```
34      if (fscanf(in, " %d %d %d\n", &u, &v, &w) != 3) break;
35      i++;
36      fprintf(out, "%d %d %d\n", i, y+v, z+w);
37    }
38    printf ("File has been created.\n");
39
40    exit (0);
41
42  }
```

B.2 derfapp.c

```
1   /*
2   ------------------------------------------------------------------------------
3    Module: DeRFAPP
4    For:    Adan Calderon
5
6    Description:
7    This program takes 3 command line arguments.
8
9    Author: Adan Calderón
10   Modification History:
11  Date       Who Modified Description
12  ------------------------------------------------------------------------------
13  Dec 04, 2013
14  ------------------------------------------------------------------------------
15  */
16  #include <stdlib.h>
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <stdio.h>
20
21  #define MAX_CHANNELS 1024
22
23
24  typedef struct
25  {
26    double v1;
27    double v2;
28  } g0line;
29
30  typedef struct
31  {
32    double v1;
33    double v2;
34    double v3;
35  } g0line2;
36
37  int main(int argc, char *argv[])
38  {
```

```
39  int i,j,k;
40  double v1,v2,v3;
41
42  FILE *in, *in2, *out;
43
44  g0line **t;
45  g0line *pool;
46  g0line *curPtr;
47
48  g0line2 *t2;
49
50  double *p3;
51
52    t = (g0line** )calloc(MAX_CHANNELS, sizeof(g0line* ));
53  pool = (g0line* )calloc(MAX_CHANNELS*MAX_CHANNELS, sizeof(g0line));
54  // Now point the pointers in the right place
55    curPtr = pool;
56    for(i = 0; i < MAX_CHANNELS; i++)
57    {
58        *(t + i) = curPtr;
59         curPtr += MAX_CHANNELS;
60    }
61
62
63  t2 = (g0line2* )calloc(MAX_CHANNELS, sizeof(g0line2));
64  p3 = (double* )calloc(MAX_CHANNELS, sizeof(double));
65
66  if (argc !=4 )
67  {
68    fprintf (stderr, "No arguments given. \n");
69    exit (1);
70  }
71
72  if ( (in=fopen(argv[1], "r")) == NULL)
73  {
74    fprintf (stderr, "Can't read %s.\n", argv[1]);
75    exit (1);
76  }
77
78  if ( (in2 = fopen (argv[2], "r")) == NULL )
79  {
80    fprintf (stderr, "Can't read %s.\n", argv[2]);
81    exit (1);
82  }
83
84  if ( (out = fopen (argv[3], "w")) == NULL )
85  {
86    fprintf (stderr, "Can't write %s.\n", argv[3]);
87    exit (1);
88  }
89
90  while (!feof(in))
```

```
91    {
92      if (fscanf(in, "%i %i %lf %lf", &i, &j, &v1, &v2) != 4) break;
93      t[i-1][j-1].v1=v1;
94      t[i-1][j-1].v2=v2;
95    }
96
97    i=0;
98    while (!feof(in2))
99    {
100     if (fscanf(in2, "%lf %lf %lf", &v1, &v2, &v3) != 3) break;
101     t2[i].v1=v1;
102     t2[i].v2=v2;
103     t2[i].v3=v3;
104     i++;
105   }
106
107
108   for (i=0; i<MAX_CHANNELS; i++)
109   {
110     for (j=0; j<MAX_CHANNELS; j++)
111     {
112       p3[j]=p3[j]+t2[i].v2*t[i][j].v2;
113     }
114   }
115
116
117   for (j=0; j<MAX_CHANNELS; j++)
118     {
119       fprintf(out,"%i %e\n",j+1, p3[j]);
120     }
121
122   free(*t);
123     free(t);
124     free(t2);
125   free(p3);
126   exit (0);
127 }
```

B.3 SpecAdder.c

```
1  /*
2  ------------------------------------------------------------------------
3   Module: SpecAdder
4   For:    Adan Calderón
5
6   Description:
7
8   Author: Adan Calderón
9   Modification History:
10 Date       Who Modified Description
```

```
11 ------------------------------------------------------------------------------
12 Feb 27, 2014
13 ------------------------------------------------------------------------------
14 */
15 #include <stdlib.h>
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <stdio.h>
19 #include <math.h>
20
21 #define MAX_LINES 1025
22 #define DETECTOR_NUMBER 0
23 #define NUMBER_FILES 156
24
25 typedef struct
26 {
27         double v1;
28         double v2;
29         double v3;
30 } line;
31
32 int main(int argc, char *argv[])
33 {
34         int i,j;
35         double v1,v2,v3;
36
37         char inputfilename[sizeof "999.out.9"];
38
39         FILE *in, *out;
40
41         line filelines[MAX_LINES];
42
43         if ( (out = fopen ("a.txt", "w")) == NULL )
44         {
45                 fprintf (stderr, "Can't write %s.\n", "a.txt");
46                 exit (1);
47         }
48         /* Zero out everything */
49         for (j=0; j<MAX_LINES; j++)
50                 {
51                         filelines[j].v1=0.0;
52                         filelines[j].v2=0.0;
53                         filelines[j].v3=0.0;
54                 }
55
56         for (j=1; j<=NUMBER_FILES; j++)
57             {
58
59                         sprintf(inputfilename, "%d.out.%d", j,DETECTOR_NUMBER);
60
61                         if ( (in = fopen (inputfilename, "r")) == NULL )
62                         {
```

```
63                                  fprintf (stderr, "Can't read %s.\n", "1.txt");
64                                  exit (1);
65                          }
66
67                          i=0;
68                          while (!feof(in))
69                          {
70                                  if (fscanf(in, "%lf %lf %lf", &v1, &v2, &v3) != 3)
                                        break;
71                                  filelines[i].v1=v1;
72                                  filelines[i].v2=filelines[i].v2+v2;
73                                  filelines[i].v3+=(v2*v3)*(v2*v3);
74                                  i++;;
75                          }
76                          fclose(in);
77              }
78
79          for (j=0; j<MAX_LINES; j++)
80                  {
81                           fprintf(out,"%e %e %e\n",filelines[j].v1,filelines[j].v2/
                               NUMBER_FILES,sqrt(filelines[j].v3)/NUMBER_FILES);
82                  }
83
84          exit (0);
85  }
```

B.4 gtotal.c

```
1   /*
2   ------------------------------------------------------------------------------
3    Module: Grand Total
4    For:    Adan Calderón
5
6    Description:
7
8    Author: Adan Calderón
9    Modification History:
10  Date       Who Modified Description
11  ------------------------------------------------------------------------------
12  Feb 27, 2014
13  ------------------------------------------------------------------------------
14  */
15
16  /*
17  Structure of the File Being READ
18  Total Counts followed by Relative Error for  22.5 Degrees
19  Total Counts followed by Relative Error for  45.0 Degrees
20  Total Counts followed by Relative Error for  67.5 Degrees
21  Total Counts followed by Relative Error for  90.0 Degrees
22  Total Counts followed by Relative Error for 112.5 Degrees
```

```
Total Counts followed by Relative Error for 135.0 Degrees
Total Counts followed by Relative Error for 157.5 Degrees
Total Counts followed by Relative Error for 180.5 Degrees

The structure repeats itself 8 times for a total of 64 Lines

Output is a single file with 64 Lines
*/
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX_LINES 64
#define NUMBER_FILES 124

typedef struct
{
        double v1;
        double v2;
        double v3;
} line;

int main(int argc, char *argv[])
{
        int i,j;
        double v1,v2,v3;

        char inputfilename[sizeof "999.TOTALS"];

        FILE *in, *out;

        line filelines[MAX_LINES];

        if ( (out = fopen ("GRAND_TOTAL.txt", "w")) == NULL )
        {
                fprintf (stderr, "Can't write %s.\n", "GRAND_TOTAL");
                exit (1);
        }
        /* Zero out everything */
        for (j=0; j<MAX_LINES; j++)
                {
                        filelines[j].v2=0.0;
                        filelines[j].v3=0.0;
                }

        for (j=1; j<=NUMBER_FILES; j++)
            {

                        sprintf(inputfilename, "%d.TOTALS", j);

```

```
75                          if ( (in = fopen (inputfilename, "r")) == NULL )
76                          {
77                                  fprintf (stderr, "Can't read %s.\n", "%d.TOTALS");
78                                  exit (1);
79                          }
80
81                          i=0;
82                          while (!feof(in))
83                          {
84                                  if (fscanf(in, "%lf %lf", &v2, &v3) != 2) break;
85                                  filelines[i].v2=filelines[i].v2+v2;
86                                  filelines[i].v3+=(v2*v3)*(v2*v3);
87                                  i++;
88                          }
89                          fclose(in);
90              }
91
92          for (j=0; j<MAX_LINES; j++)
93                  {
94                          fprintf(out,"%e %e\n",filelines[j].v2/NUMBER_FILES,sqrt(
                                filelines[j].v3)/NUMBER_FILES);
95                  }
96          exit (0);
97  }
```

B.5 pixiedust.c

```
1   /*
2   ------------------------------------------------------------------------------
3    Module: PixieDust
4    For:    Adan Calderón
5
6    Description:
7    This program takes two command line arguments.  The first is the binary output
8    file from the XIA pixie.  The second is a destination file.
9
10   Usage: pixiedust filename.bin output.txt
11
12   Author: Adan Calderón
13   Modification History:
14  Date       Who Modified Description
15  ------------------------------------------------------------------------------
16  April 29 2012
17  ------------------------------------------------------------------------------
18  */
19  #include <stdlib.h>
20  #include <stdio.h>
21
22  #define BYTETOBINARYPATTERN "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d"
23  #define BYTETOBINARY(byte)    \
```

```
  (byte & 0x8000 ? 1 : 0), \
  (byte & 0x4000 ? 1 : 0), \
    (byte & 0x2000 ? 1 : 0), \
    (byte & 0x1000 ? 1 : 0), \
  (byte & 0x0800 ? 1 : 0), \
  (byte & 0x0400 ? 1 : 0), \
  (byte & 0x0200 ? 1 : 0), \
  (byte & 0x0100 ? 1 : 0), \
  (byte & 0x0080 ? 1 : 0), \
  (byte & 0x0040 ? 1 : 0), \
  (byte & 0x0020 ? 1 : 0), \
  (byte & 0x0010 ? 1 : 0), \
  (byte & 0x0008 ? 1 : 0), \
  (byte & 0x0004 ? 1 : 0), \
  (byte & 0x0002 ? 1 : 0), \
  (byte & 0x0001 ? 1 : 0)

int main(int argc, char *argv[])
{
  struct BufferHeader
  {
  unsigned short int BUF_NDATA;
  unsigned short int BUF_MODNUM;
  unsigned short int BUF_FORMAT;
  unsigned short int BUF_TIMEHI;
  unsigned short int BUF_TIMEMI;
  unsigned short int BUF_TIMELO;
    };

  struct EventHeader
  {
    unsigned short int EVT_PATTERN;
    unsigned short int EVT_TIMEHI;
    unsigned short int EVT_TIMELO;
  };

  struct ChannelHeader9
  {
    unsigned short int CHAN_NDATA;
    unsigned short int CHAN_TRIGTIME;
    unsigned short int CHAN_ENERGY;
    unsigned short int CHAN_XIAPSA;
    unsigned short int CHAN_USERPSA;
    unsigned short int Unused0;
    unsigned short int Unused1;
    unsigned short int Unused2;
    unsigned short int CHAN_REALTIMEHI;
  };

  struct ChannelHeader4
  {
    unsigned short int CHAN_TRIGTIME;
```

```
76      unsigned short int CHAN_ENERGY;
77      unsigned short int CHAN_XIAPSA;
78      unsigned short int CHAN_USERPSA;
79    };
80
81    struct ChannelHeader2
82    {
83      unsigned short int CHAN_TRIGTIME;
84      unsigned short int CHAN_ENERGY;
85    };
86
87
88    unsigned short int CHANHEADLEN;
89    unsigned short int RUNTASK;
90    unsigned short int N_WAVE_DATA;
91    unsigned short int temp;
92    unsigned short int BUFFERBYTES;
93    unsigned short int BUFFERNUMBER;
94    unsigned int EVENTNUMBER;
95
96    struct BufferHeader CurrentBufferHeader;
97    struct EventHeader eventHeader;
98    struct ChannelHeader9 channelHeader9;
99    struct ChannelHeader4 channelHeader4;
100   struct ChannelHeader2 channelHeader2;
101
102   FILE *in, *out;
103
104   if (argc !=3 )
105   {
106     fprintf (stderr, "No arguments given. \n");
107     exit (1);
108   }
109
110   if ( (in=fopen(argv[1], "rb")) == NULL)
111   {
112     fprintf (stderr, "Can't read %s.\n", argv[1]);
113     exit (1);
114   }
115
116   if ( (out = fopen (argv[2], "w")) == NULL )
117   {
118     fprintf (stderr, "Can't write %s.\n", argv[2]);
119     exit (1);
120   }
121
122   EVENTNUMBER=0;
123   BUFFERNUMBER=0;
124
125   /*
126   While not end of file keep reading from it
127   */
```

```
128   while (fread(&CurrentBufferHeader,12,1,in)!=0)
129   {
130
131     /*
132     Begin by reading the first 12 bytes of the file
133     This information will be used to determin the RUNTASK
134     */
135
136     /*
137     printf("The Current Buffer is %hu \n", BUFFERNUMBER);
138     printf("Number of words in this Buffer %hu \n", CurrentBufferHeader.BUF_NDATA)
          ;
139     printf("Run start time, high word %hu \n", CurrentBufferHeader.BUF_TIMEHI);
140     printf("Run start time, middle word %hu \n", CurrentBufferHeader.BUF_TIMEMI);
141     printf("Run start time, low word %hu \n", CurrentBufferHeader.BUF_TIMELO);
142     */
143
144     /*
145     Calculate how many bytes remain in the current buffer.
146     Since the BUF_NDATA is the amount of 16-Bit words in the
147     entire buffer, we subtract the header size of 6 words.
148     We then multiply this by 2 to get the total remaining bytes.
149     */
150     BUFFERBYTES=(CurrentBufferHeader.BUF_NDATA-6)*2;
151
152     /*
153     RUNTASK=FORMAT DESCRIPTOR - AN OFFSET
154     The Pixie-500 500Mhz Version has OFFSET 0x4000
155       with TimeStamps in units of 2 ns and increments of 8ns
156
157     The Pixie-500 400Mhz Version has OFFSET 0x5000
158       with TimeStamps in unites of 2.5 ns and increments of 13.33ns
159
160     The Pixie-4 has OFFSET 0x2000
161
162     */
163     RUNTASK=CurrentBufferHeader.BUF_FORMAT-0x4000;
164     /*
165     printf("The RUNTASK IS %hu \n", RUNTASK);
166     */
167     {
168       /*
169       Determine Channel Header Type from RUNTASK
170       */
171       if (RUNTASK==256)
172       {
173         CHANHEADLEN=9;
174       }
175
176       if (RUNTASK==257)
177       {
178         CHANHEADLEN=9;
```

```
179         }
180
181         if (RUNTASK==258)
182         {
183           CHANHEADLEN=4;
184         }
185
186         if (RUNTASK==259)
187         {
188           CHANHEADLEN=2;
189         }
190       }
191
192       //EVENTNUMBER=0;
193
194       /*
195       While there are still bytes in the buffer keep reading
196       */
197       while (BUFFERBYTES>=1)
198       {
199         fread(&eventHeader,6,1,in);
200         BUFFERBYTES=BUFFERBYTES-6;
201         fprintf (out, "      The current event number is %u \n", EVENTNUMBER);
202         fprintf (out, "EVENT PATTERN "BYTETOBINARYPATTERN"\n", BYTETOBINARY(
              eventHeader.EVT_PATTERN));
203
204         /*
205         Setup for channels that actually got used
206         */
207         fprintf(out,"Event time, high word %hu \n", eventHeader.EVT_TIMEHI);
208         fprintf(out,"Event time, low word %hu \n", eventHeader.EVT_TIMELO);
209
210         if (CHANHEADLEN==9)
211         {
212           /*
213           Did Channel 0 (Detector 1) get DATA?
214           */
215
216           if ((eventHeader.EVT_PATTERN & 1) == 1)
217           {
218             fread(&channelHeader9,CHANHEADLEN*2,1,in);
219               BUFFERBYTES=BUFFERBYTES-(CHANHEADLEN*2);
220             N_WAVE_DATA=channelHeader9.CHAN_NDATA-CHANHEADLEN;
221             fprintf(out,"Channel 0\n");
222             fprintf(out,"Fast trigger time %hu\n",channelHeader9.CHAN_TRIGTIME);
223             fprintf(out,"Energy %hu\n",channelHeader9.CHAN_ENERGY);
224             fprintf(out,"High word of the real time %hu\n",channelHeader9.
                  CHAN_REALTIMEHI);
225             while (N_WAVE_DATA>=1)
226             {
227               fread(&temp,2,1,in);
228               fprintf(out,"%hu\n",temp);
```

```
229             BUFFERBYTES=BUFFERBYTES-2;
230             N_WAVE_DATA--;
231           }
232         }
233
234         /*
235         Did Channel 1 (Detector 2) get DATA?
236         */
237         if ((eventHeader.EVT_PATTERN & 2) == 2)
238         {
239           fread(&channelHeader9,CHANHEADLEN*2,1,in);
240             BUFFERBYTES=BUFFERBYTES-(CHANHEADLEN*2);
241           N_WAVE_DATA=channelHeader9.CHAN_NDATA-CHANHEADLEN;
242           fprintf(out,"Channel 1\n");
243           fprintf(out,"Fast trigger time %hu\n",channelHeader9.CHAN_TRIGTIME);
244           fprintf(out,"Energy %hu\n",channelHeader9.CHAN_ENERGY);
245           fprintf(out,"High word of the real time %hu\n",channelHeader9.
                  CHAN_REALTIMEHI);
246           while (N_WAVE_DATA>=1)
247           {
248             fread(&temp,2,1,in);
249             fprintf(out,"%hu\n",temp);
250             BUFFERBYTES=BUFFERBYTES-2;
251             N_WAVE_DATA--;
252           }
253         }
254
255         /*
256         Did Channel 2 (Detector 3) get DATA?
257         */
258         if ((eventHeader.EVT_PATTERN & 4) == 4)
259         {
260           fread(&channelHeader9,CHANHEADLEN*2,1,in);
261             BUFFERBYTES=BUFFERBYTES-(CHANHEADLEN*2);
262           N_WAVE_DATA=channelHeader9.CHAN_NDATA-CHANHEADLEN;
263           fprintf(out,"Channel 2\n");
264           fprintf(out,"Fast trigger time %hu\n",channelHeader9.CHAN_TRIGTIME);
265           fprintf(out,"Energy %hu\n",channelHeader9.CHAN_ENERGY);
266           fprintf(out,"High word of the real time %hu\n",channelHeader9.
                  CHAN_REALTIMEHI);
267           while (N_WAVE_DATA>=1)
268           {
269             fread(&temp,2,1,in);
270             fprintf(out,"%hu\n",temp);
271             BUFFERBYTES=BUFFERBYTES-2;
272             N_WAVE_DATA--;
273           }
274         }
275
276         /*
277         Did Channel 3 (Detector 4) get DATA?
278         */
```

```
279           if ((eventHeader.EVT_PATTERN & 8) == 8)
280           {
281             fread(&channelHeader9,CHANHEADLEN*2,1,in);
282               BUFFERBYTES=BUFFERBYTES-(CHANHEADLEN*2);
283             N_WAVE_DATA=channelHeader9.CHAN_NDATA-CHANHEADLEN;
284             fprintf(out,"Channel 3\n");
285             fprintf(out,"Fast trigger time %hu\n",channelHeader9.CHAN_TRIGTIME);
286             fprintf(out,"Energy %hu\n",channelHeader9.CHAN_ENERGY);
287             fprintf(out,"High word of the real time %hu\n",channelHeader9.
                    CHAN_REALTIMEHI);
288             while (N_WAVE_DATA>=1)
289             {
290               fread(&temp,2,1,in);
291               fprintf(out,"%hu\n",temp);
292               BUFFERBYTES=BUFFERBYTES-2;
293               N_WAVE_DATA--;
294             }
295           }
296         }
297
298         if (CHANHEADLEN==4)
299         {
300           fread(&channelHeader4,CHANHEADLEN*2,1,in);
301         }
302
303         if (CHANHEADLEN==2)
304         {
305           fread(&channelHeader2,CHANHEADLEN*2,1,in);
306         }
307        EVENTNUMBER++;
308       }
309      BUFFERNUMBER++;
310    }
311    printf("\n");
312    //printf ("File has been created.\n");
313    exit (0);
314  }
```

B.6 convo.c

```
1  /*
2  ------------------------------------------------------------------------
3   Module: Comvo Adder for PMFs
4   For:    Adan Calderón
5
6   Description:
7
8   Author: Adan Calderón
9   Modification History:
10 Date       Who Modified Description
```

```
11 ------------------------------------------------------------------------
12 Feb 27, 2014
13 ------------------------------------------------------------------------
14 */
15 #include <stdlib.h>
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <stdio.h>
19 #include <math.h>
20
21 #define MAX_LINES 1025
22
23 typedef struct
24 {
25         double v1;
26       double v2;
27         double v3;
28 } line;
29
30 int main(int argc, char *argv[])
31 {
32         int i,j;
33         double v1,v2,v3;
34         FILE *in, *in2, *out;
35         line pmf_in[MAX_LINES];
36     line pmf_in2[MAX_LINES];
37     line pmf_out[MAX_LINES];
38
39     if (argc !=4 )
40       {
41       fprintf (stderr, "Correct ussage is:\n");
42         fprintf (stderr, "%s inputfile1 inputfile2 outputfile n \n",argv[0]);
43         fprintf (stderr, " \n");
44         exit (1);
45       }
46
47     if ( (in=fopen(argv[1], "r")) == NULL)
48       {
49       fprintf (stderr, "Can't read %s.\n", argv[1]);
50         exit (1);
51       }
52
53     if ( (in2=fopen(argv[2], "r")) == NULL)
54       {
55       fprintf (stderr, "Can't read %s.\n", argv[1]);
56         exit (1);
57       }
58
59     if ( (out = fopen (argv[3], "w")) == NULL )
60       {
61       fprintf (stderr, "Can't write %s.\n", argv[2]);
62         exit (1);
```

```
63        }
64
65          /* Zero out everything */
66          for (j=0; j<MAX_LINES; j++)
67                  {
68                 pmf_in[j].v1=0.0;
69                          pmf_in[j].v2=0.0;
70                          pmf_in[j].v3=0.0;
71
72               pmf_in2[j].v1=0.0;
73               pmf_in2[j].v2=0.0;
74                          pmf_in2[j].v3=0.0;
75
76               pmf_out[j].v1=0.0;
77               pmf_out[j].v2=0.0;
78                          pmf_out[j].v3=0.0;
79                  }
80      i=0;
81      while (!feof(in))
82              {
83            if (fscanf(in, "%lf %lf %lf", &v1, &v2, &v3) != 3) break;
84            pmf_in[i].v1=v1;
85            pmf_in[i].v2=v2;
86            pmf_in[i].v3=v3;
87                      i++;
88              }
89          fclose(in);
90
91      i=0;
92      while (!feof(in2))
93              {
94            if (fscanf(in2, "%lf %lf %lf", &v1, &v2, &v3) != 3) break;
95            pmf_in2[i].v1=v1;
96                      pmf_in2[i].v2=v2;
97            pmf_in2[i].v3=v3;
98                      i++;
99              }
100         fclose(in2);
101
102         for (j=0; j<=MAX_LINES; j++)
103             {
104         pmf_out[j].v1=pmf_in[j].v1;
105         for (i=0; i<=j; i++)
106                 {
107              pmf_out[j].v2=pmf_out[j].v2 + pmf_in[i].v2 * pmf_in2[j-i].v2;
108         }
109         }
110
111     for (j=0; j<MAX_LINES; j++)
112             {
113         fprintf(out, "%e %e\n", pmf_out[j].v1, pmf_out[j].v2);
114       }
```

```
115
116     fclose(out);
117         exit (0);
118 }
```

B.7 chnconvert.f90

```
1  !*****************************************************************************
2  !
3  !  PROGRAM: CHNCONVERT
4  !
5  !  PURPOSE:  To converts ORTEC .CHN file format to an ASCII format.
6  !
7  !
8  ! Record of revisons:
9  !      Date          Programmer          Description of change
10 !      ====          ==========          =====================
11 !  Feb. 19 2011      A. F. Calderón      Initial write of program
12 !*****************************************************************************
13     program CHNCONVERT
14
15     implicit none
16
17     INTEGER, PARAMETER :: I1B=SELECTED_INT_KIND(2)
18     INTEGER, PARAMETER :: I2B=SELECTED_INT_KIND(4)
19     INTEGER, PARAMETER :: I4B=SELECTED_INT_KIND(9)
20     INTEGER, PARAMETER :: I8B=SELECTED_INT_KIND(18)
21     INTEGER, PARAMETER :: R1B=SELECTED_REAL_KIND(r=2)
22     INTEGER, PARAMETER :: R2B=SELECTED_REAL_KIND(r=4)
23     INTEGER, PARAMETER :: R4B=SELECTED_REAL_KIND(r=9)
24     INTEGER, PARAMETER :: R8B=SELECTED_REAL_KIND(r=18)
25     INTEGER, PARAMETER :: ONEBYTEOFFSET=256          ! USED TO CONVERT TO UNSIGNED
26     INTEGER, PARAMETER :: TWOBYTEOFFSET=65536        ! USED TO CONVERT TO UNSIGNED
27     INTEGER, PARAMETER :: FOURBYTEOFFSET=4294967296 ! USED TO CONVERT TO UNSIGNED
28
29     ! Variables
30     CHARACTER,ALLOCATABLE,DIMENSION(:) :: a          ! Data array
31     INTEGER (KIND=I8B),ALLOCATABLE,DIMENSION(:) :: CHANNEL
32     CHARACTER (len=20) :: filenameI                  ! Input data file name
33     CHARACTER (len=20) :: filenameO                  ! Output data file name
34     INTEGER (KIND=I4B) :: FOURBYTES
35     INTEGER (KIND=I1B) :: ONEBYTE
36     INTEGER (KIND=I1B) :: FOURBYTE_ARRAY(4)
37     INTEGER (KIND=I1B) :: TWOBYTE_ARRAY(2)
38     INTEGER (KIND=I1B) :: THREEBYTE_ARRAY(3)
39     INTEGER (KIND=I1B) :: TEMP_DESC_ARRAY(63)
40     INTEGER (KIND=I2B) :: HEADER_CHECK
41     INTEGER (KIND=I2B) :: MCA_DET_NUMBER
42     INTEGER (KIND=I2B) :: SEGMENT_NUMBER
43     CHARACTER (LEN=2)  :: SECONDS
```

```
44      INTEGER (KIND=I4B) :: REAL_TIME
45      INTEGER (KIND=I4B) :: LIVE_TIME
46      CHARACTER (LEN=2)  :: DAY
47      CHARACTER (LEN=3)  :: MONTH
48      CHARACTER (LEN=2)  :: YEAR
49      CHARACTER          :: Y2K_CHECK
50      CHARACTER (LEN=2)  :: START_HOUR
51      CHARACTER (LEN=2)  :: START_MINS
52      INTEGER (KIND=I2B) :: CHANNEL_OFFSET
53      INTEGER (KIND=I2B) :: TEMP_NUM_CHAN
54      INTEGER (KIND=I2B) :: TEST_NEG_102_TEMP
55      INTEGER (KIND=I4B) :: TEST_NEG_102
56      INTEGER (KIND=I4B) :: NUMBER_OF_CHANNELS
57      INTEGER (KIND=I4B) :: PRESENT_CHANNEL
58      INTEGER (KIND=I4B) :: TEMP_CHANNEL
59      INTEGER :: YEAR_4DIGITS
60      INTEGER  :: status                  ! Status: 0 for sucess
61      INTEGER  :: nvals = 0               ! Number of values to process
62      INTEGER  :: mypos,i,j
63      CHARACTER :: temp
64
65      REAL (KIND=R4B) :: ENERGY_CAL_INT !ENERGY CALIBRATION INTERCEPT       0.0 for
            uncalibrated spectrum
66      REAL (KIND=R4B) :: ENERGY_CAL_SLP !ENERGY CALIBRATION SLOPE           1.0 for
            uncalibrated spectrum
67      REAL (KIND=R4B) :: ENERGY_CAL_QUD !ENERGY CALIBRATION QUADRATIC TERM  0.0 for
            uncalibrated spectrum
68      REAL (KIND=R4B) :: PEAK_CAL_INT   !PEAK SHAPE CALIBRATION INTERCEPT   1.0 for
            uncalibrated spectrum
69      REAL (KIND=R4B) :: PEAK_CAL_SLP   !PEAK SHAPE CALIBRATION SLOPE       0.0 for
            uncalibrated spectrum
70      REAL (KIND=R4B) :: PEAK_CAL_QUD
71      INTEGER (KIND=I1B) :: DET_DESC_LEN !DETECTOR DESCRIPTION LENGTH
72      CHARACTER (LEN=63) :: DET_DESCRIPTION
73      INTEGER (KIND=I1B) :: SAMP_DESC_LEN !SAMPLE DESCRIPTION LENGTH
74      CHARACTER (LEN=63) :: SAMP_DESCRIPTION
75
76      ! Body of CHNCONVERT
77      WRITE (*,1000)
78      1000 FORMAT (1X,'Enter the file name to be read:')
79      READ (*,'(A20)') filenameI
80
81      ! Open input data file.  Status is OLD because the input data must
82      ! already exist.
83      OPEN ( UNIT=9, FILE=filenameI, STATUS='OLD', ACCESS='STREAM', ACTION='READ',
            FORM='UNFORMATTED', convert='LITTLE_ENDIAN', IOSTAT=status )
84
85      ! Was the OPEN successful?
86      fileopen: IF ( status == 0 ) THEN       ! Open successful
87         ! The file was opened successfully, so read the data to find
88         ! out how many values are in the file and allocate the
89         ! required space.
```

```
90          mypos=1
91          DO
92             READ (9, POS=mypos, IOSTAT=status) temp    ! Get value
93             IF ( status /= 0 ) EXIT            ! Exit on end of data
94             nvals = nvals + 1                  ! Bump count
95             mypos = mypos + 1
96          ENDDO
97
98          ! Allocate memory
99          WRITE (*,*) ' Allocating a: size = ', nvals
100         ALLOCATE ( a(nvals), STAT=status )    ! Allocate memory
101         allocate_ok:  IF ( status == 0 ) THEN
102
103         ! Was allocation successful?  If so, rewind file, read in
104         ! data, and process it.
105         ! Now read in the data.  We know that there are enough
106         ! values to fill the array.
107            DO mypos=1,nvals
108               READ (9, POS=mypos, IOSTAT=status) a(mypos)    ! Get value
109               IF ( status /= 0 ) EXIT            ! Exit on end of data
110            ENDDO
111         ENDIF allocate_ok
112
113         CLOSE(9)
114
115         TWOBYTE_ARRAY(1)=TRANSFER(a(3),ONEBYTE)
116         TWOBYTE_ARRAY(2)=TRANSFER(a(4),ONEBYTE)
117         MCA_DET_NUMBER=TRANSFER(TWOBYTE_ARRAY,MCA_DET_NUMBER)
118
119         TWOBYTE_ARRAY(1)=TRANSFER(a(5),ONEBYTE)
120         TWOBYTE_ARRAY(2)=TRANSFER(a(6),ONEBYTE)
121         SEGMENT_NUMBER=TRANSFER(TWOBYTE_ARRAY,SEGMENT_NUMBER)
122
123         TWOBYTE_ARRAY(1)=TRANSFER(a(7),ONEBYTE)
124         TWOBYTE_ARRAY(2)=TRANSFER(a(8),ONEBYTE)
125         SECONDS=TRANSFER(TWOBYTE_ARRAY,SECONDS)
126
127         FOURBYTE_ARRAY(1)=TRANSFER(a(9),ONEBYTE)
128         FOURBYTE_ARRAY(2)=TRANSFER(a(10),ONEBYTE)
129         FOURBYTE_ARRAY(3)=TRANSFER(a(11),ONEBYTE)
130         FOURBYTE_ARRAY(4)=TRANSFER(a(12),ONEBYTE)
131         REAL_TIME=TRANSFER(FOURBYTE_ARRAY,REAL_TIME)
132
133         FOURBYTE_ARRAY(1)=TRANSFER(a(13),ONEBYTE)
134         FOURBYTE_ARRAY(2)=TRANSFER(a(14),ONEBYTE)
135         FOURBYTE_ARRAY(3)=TRANSFER(a(15),ONEBYTE)
136         FOURBYTE_ARRAY(4)=TRANSFER(a(16),ONEBYTE)
137         LIVE_TIME=TRANSFER(FOURBYTE_ARRAY,LIVE_TIME)
138
139         TWOBYTE_ARRAY(1)=TRANSFER(a(17),ONEBYTE)
140         TWOBYTE_ARRAY(2)=TRANSFER(a(18),ONEBYTE)
141         DAY=TRANSFER(TWOBYTE_ARRAY,DAY)
```

```
142
143         THREEBYTE_ARRAY(1)=TRANSFER(a(19),ONEBYTE)
144         THREEBYTE_ARRAY(2)=TRANSFER(a(20),ONEBYTE)
145         THREEBYTE_ARRAY(3)=TRANSFER(a(21),ONEBYTE)
146         MONTH=TRANSFER(THREEBYTE_ARRAY,MONTH)
147
148         TWOBYTE_ARRAY(1)=TRANSFER(a(22),ONEBYTE)
149         TWOBYTE_ARRAY(2)=TRANSFER(a(23),ONEBYTE)
150         YEAR=TRANSFER(TWOBYTE_ARRAY,YEAR)
151
152         Y2K_CHECK=TRANSFER(a(24),Y2K_CHECK)
153
154         TWOBYTE_ARRAY(1)=TRANSFER(a(25),ONEBYTE)
155         TWOBYTE_ARRAY(2)=TRANSFER(a(26),ONEBYTE)
156         START_HOUR=TRANSFER(TWOBYTE_ARRAY,START_HOUR)
157
158         TWOBYTE_ARRAY(1)=TRANSFER(a(27),ONEBYTE)
159         TWOBYTE_ARRAY(2)=TRANSFER(a(28),ONEBYTE)
160         START_MINS=TRANSFER(TWOBYTE_ARRAY,START_MINS)
161
162         TWOBYTE_ARRAY(1)=TRANSFER(a(29),ONEBYTE)
163         TWOBYTE_ARRAY(2)=TRANSFER(a(30),ONEBYTE)
164         CHANNEL_OFFSET=TRANSFER(TWOBYTE_ARRAY,CHANNEL_OFFSET)
165
166         TWOBYTE_ARRAY(1)=TRANSFER(a(31),ONEBYTE)
167         TWOBYTE_ARRAY(2)=TRANSFER(a(32),ONEBYTE)
168         TEMP_NUM_CHAN=TRANSFER(TWOBYTE_ARRAY,TEMP_NUM_CHAN)
169         NUMBER_OF_CHANNELS=TEMP_NUM_CHAN+TWOBYTEOFFSET
170
171         WRITE (*,*) ' Allocating Channel(s): size = ', NUMBER_OF_CHANNELS
172         ALLOCATE ( CHANNEL(NUMBER_OF_CHANNELS), STAT=status )    ! Allocate memory
173         allocate2_ok:  IF ( status == 0 ) THEN
174
175         ! Was allocation successful?  If so, rewind file, read in
176         ! data, and process it.
177         ! Now read in the data.  We know that there are enough
178         ! values to fill the array.
179
180         PRESENT_CHANNEL=1
181            DO mypos=1,NUMBER_OF_CHANNELS*4,4
182               FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+32),ONEBYTE)
183               FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+33),ONEBYTE)
184               FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+34),ONEBYTE)
185               FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+35),ONEBYTE)
186               TEMP_CHANNEL=TRANSFER(FOURBYTE_ARRAY,TEMP_CHANNEL)
187               !WRITE(*,*) TEMP_CHANNEL
188               CHANNEL(PRESENT_CHANNEL)=TEMP_CHANNEL+FOURBYTEOFFSET
189               PRESENT_CHANNEL=PRESENT_CHANNEL+1
190            ENDDO
191
192         ENDIF allocate2_ok
193
```

```
194
195
196         mypos=32+NUMBER_OF_CHANNELS*4+1
197
198         TWOBYTE_ARRAY(1)=TRANSFER(a(mypos),ONEBYTE)
199         TWOBYTE_ARRAY(2)=TRANSFER(a(mypos+1),ONEBYTE)
200         TEST_NEG_102_TEMP=TRANSFER(TWOBYTE_ARRAY,TEST_NEG_102_TEMP)
201         TEST_NEG_102=TEST_NEG_102_TEMP
202         WRITE(*,*) 'NEGATIVE 102 TEST :',TEST_NEG_102
203
204         FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+4),ONEBYTE)
205         FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+5),ONEBYTE)
206         FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+6),ONEBYTE)
207         FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+7),ONEBYTE)
208         ENERGY_CAL_INT=TRANSFER(FOURBYTE_ARRAY,ENERGY_CAL_INT)
209
210         FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+8),ONEBYTE)
211         FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+9),ONEBYTE)
212         FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+10),ONEBYTE)
213         FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+11),ONEBYTE)
214         ENERGY_CAL_SLP=TRANSFER(FOURBYTE_ARRAY,ENERGY_CAL_SLP)
215
216         FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+12),ONEBYTE)
217         FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+13),ONEBYTE)
218         FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+14),ONEBYTE)
219         FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+15),ONEBYTE)
220         ENERGY_CAL_QUD=TRANSFER(FOURBYTE_ARRAY,ENERGY_CAL_QUD)
221
222         FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+16),ONEBYTE)
223         FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+17),ONEBYTE)
224         FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+18),ONEBYTE)
225         FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+19),ONEBYTE)
226         PEAK_CAL_INT=TRANSFER(FOURBYTE_ARRAY,PEAK_CAL_INT)
227
228         FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+20),ONEBYTE)
229         FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+21),ONEBYTE)
230         FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+22),ONEBYTE)
231         FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+23),ONEBYTE)
232         PEAK_CAL_SLP=TRANSFER(FOURBYTE_ARRAY,PEAK_CAL_SLP)
233
234         FOURBYTE_ARRAY(1)=TRANSFER(a(mypos+24),ONEBYTE)
235         FOURBYTE_ARRAY(2)=TRANSFER(a(mypos+25),ONEBYTE)
236         FOURBYTE_ARRAY(3)=TRANSFER(a(mypos+26),ONEBYTE)
237         FOURBYTE_ARRAY(4)=TRANSFER(a(mypos+27),ONEBYTE)
238         PEAK_CAL_QUD=TRANSFER(FOURBYTE_ARRAY,PEAK_CAL_QUD)
239
240         DET_DESC_LEN=TRANSFER(a(mypos+256),ONEBYTE)
241
242         j=1
243         DO i=mypos+257,mypos+257+62
244            TEMP_DESC_ARRAY(j)=TRANSFER(a(i),ONEBYTE)
245            j=j+1
```

```
246         ENDDO
247         DET_DESCRIPTION=TRANSFER(TEMP_DESC_ARRAY,DET_DESCRIPTION)
248
249         SAMP_DESC_LEN=TRANSFER(a(mypos+320),ONEBYTE)
250
251         j=1
252         DO i=mypos+321,mypos+321+62
253           TEMP_DESC_ARRAY(j)=TRANSFER(a(i),ONEBYTE)
254           j=j+1
255         ENDDO
256         SAMP_DESCRIPTION=TRANSFER(TEMP_DESC_ARRAY,SAMP_DESCRIPTION)
257
258
259
260
261     ELSE fileopen
262        ! Else file open failed.  Tell user.
263        WRITE (*,1050) status
264        1050 FORMAT (1X,'File open failed--status = ', I6)
265     ENDIF fileopen
266     write(*,*) 'DEBUG TEST'
267
268     IF (Y2K_CHECK=='1') THEN
269       read (YEAR,'(I4)') YEAR_4DIGITS
270       YEAR_4DIGITS=2000+YEAR_4DIGITS
271     ELSE
272       read (YEAR,'(I4)') YEAR_4DIGITS
273       YEAR_4DIGITS=1900+YEAR_4DIGITS
274     ENDIF
275
276     OPEN (UNIT = 7, FILE="OUTPUT.TXT",STATUS="UNKNOWN")
277     WRITE(7,*) 'Date                  : ',MONTH,' ',DAY, ' ',YEAR_4DIGITS
278     WRITE(7,*) 'Start Time            : ',START_HOUR,':',START_MINS
279     WRITE(7,*) 'Dectector/MCA Number  : ',MCA_DET_NUMBER
280     WRITE(7,*) 'Segment number        : ',SEGMENT_NUMBER
281     WRITE(7,*) 'Number of Channels    : ',NUMBER_OF_CHANNELS
282     WRITE(7,*) 'Channel DATA Offset   : ',CHANNEL_OFFSET
283     WRITE(7,*) 'Detector Description  : ',DET_DESCRIPTION
284     WRITE(7,*) 'Sample Description    : ',SAMP_DESCRIPTION
285     WRITE(7,*) 'Live Time             : ',LIVE_TIME
286     WRITE(7,*) 'Real Time             : ',REAL_TIME
287     WRITE(7,*) 'Energy Cal Intercept  : ',ENERGY_CAL_INT
288     WRITE(7,*) 'Energy Cal Slope      : ',ENERGY_CAL_SLP
289     WRITE(7,*) 'Energy Cal Quadratic  : ',ENERGY_CAL_QUD
290     WRITE(7,*) 'Peak Cal Intercept    : ',PEAK_CAL_INT
291     WRITE(7,*) 'Peak Cal Slope        : ',PEAK_CAL_SLP
292     WRITE(7,*) 'Peak Cal Quadratic    : ',PEAK_CAL_QUD
293     WRITE(7,*)
294
295
296     DO PRESENT_CHANNEL=1,NUMBER_OF_CHANNELS
297       WRITE(7,*) PRESENT_CHANNEL,CHANNEL(PRESENT_CHANNEL)
```

```
298     ENDDO
299     ! Deallocate the array now that we are done.
300     DEALLOCATE ( a, STAT=status )
301     DEALLOCATE ( CHANNEL, STAT=status )
302     end program CHNCONVERT
```

B.8 BASH Script to extract tallys from MCNP

```
1  #!/bin/sh
2  startlines=(`grep -n "cell  [0-9]" $1|cut -d ':' -f1`)
3  endlines=(`grep -n "      total      [0-9]\.[0-9]" $1|cut -d ':' -f1`)
4  for i in "${!startlines[@]}"
5  do
6    startline=`expr ${startlines[i]} + 2`
7    endline=`expr ${endlines[i]} - 1`
8    `sed -e "$startline,$endline!d" $1 > out.$i`
9  done
```

B.9 BASH Script to run MCNP-CP in Parallel on CEAR Cluster

```
1  #!/bin/sh
2  # ##############################################################################
3
4          NAME=${0##*/} ## Get the name of the script without its path
5          HTML="Runs MCNP-CP on the CEAR CLUSTER in Distributed manner"
6       PURPOSE="To distribute MCNP-CP jobs on CEAR Cluster"
7      SYNOPSIS="$NAME -t tmplt.in [-s SeedsFile] [-n NumberOfNodes] -o <output_dir>
           "
8      REQUIRES="standard GNU commands and a template input file"
9       VERSION="1.0"
10         DATE="2013-09-12; last update: 2014-02-26"
11       AUTHOR="Adan F. Calderon Jr. <adancalderon@gmail.com>"
12          URL="www.cearonline.com"
13     CATEGORY="file"
14     PLATFORM="Linux"
15        SHELL="bash"
16   DISTRIBUTE="yes"
17
18 # ##############################################################################
19 # This program is distributed under the terms of the GNU General Public License
20 # Version 2
21 # HISTORY:
22 # 2013-09-12 v1.0 - Initial Version
23 #
24 usage () {
```

```
25 printf >&2 "\n$NAME_ $VERSION_ - $PURPOSE_
26
27 Traditionally one would run the serial version of MCNP-CP by executing the
28 following:
29
30          mcnp-cp inp=infile out=outfile dumn1=logfile
31
32 After a successful run, the files outfile and logfile would be created.
33
34 This script will help run multiple instances of such a command line on multiple
35 computing nodes. $NAME_ will create a directory structure in the specified
36 output directory. This structure consists of multiple subdirectories with a
37 unique numerical value . There will be as many subdirectories as there are total
38 jobs submitted. For Instance if you specify -n 36, there will be a total of 144
39 subdirectories created because 36 times 4 is 144. The 4 comes from the number of
40 processors per node.
41
42 With 36 nodes specified, you will have
43
44 node100:
45 /home/$USER/local/1
46 /home/$USER/local/2
47 /home/$USER/local/3
48 /home/$USER/local/4
49 node102:
50 /home/$USER/local/5
51 /home/$USER/local/6
52 /home/$USER/local/7
53 /home/$USER/local/8
54 and so forth until
55 .
56 .
57 .
58 node140:
59 /home/$USER/local/141
60 /home/$USER/local/142
61 /home/$USER/local/143
62 /home/$USER/local/144
63
64 In each of these numerical subdirectories one will find output and log files.
65 The input file that is used for each of these runs will have a different random
66 seed. Please make certain that your input file template has
67 the following line at the end:
68
69 c rand seed RNSEED
70
71 \nUsage: $SYNOPSIS_
72
73 Requires: $REQUIRES_
74
75 Options:
76
```

```
77      -t, <TemplateInputFile>, A template input file for MCNP-CP
78          this filenmae should be 8 characters or less
79      -n, <NumberOfNodes>, Number of computing nodes to use. Only needed if less
80          then the maximum number of available nodes is needed. At present time
81          the maximum number is 40.  There are 40 32-Bit nodes.
82      -s, <SeedsFile>, Optional text file with a seed per line to use on each run.
83          The file must contain at least as many seeds as jobs that will be
                submitted.
84          For example if you are using 36 nodes with 4 processors per node, then
                this
85          file must have at least 144 seeds.
86          If this is not specified then and odd seed is randomly created for each
                job.
87      -o, <OutputDirectory>, path to where to create the output folders.  This will
88          more then likely be /home/$USER/local or ~/local
89      -h, usage and options (this help)
90      -l, see this script
91
92  Examples:
93  $NAME_ -t template.inp -o ~/local
94  $NAME_ -t sampledeck.inp -s MySeeds.txt -n 36 /home/$USER/local
95  \n"
96  exit 1
97  }
98
99  # args check
100 [ $# -eq 0 ] && { echo >&2 missing argument, type $NAME_ -h for help; exit 1; }
101
102 trap "exit 1" 1 2 3 15
103
104 # var init
105 TemplateInputFile=
106 NumberOfNodes=
107 SeedsFile=
108 OutputDirectory=
109
110 while getopts hlns:t:o: options; do
111
112     case "$options" in
113         t) TemplateInputFile="$OPTARG" ;;
114         n) NumberOfNodes="$OPTARG" ;;
115         s) SeedsFile="$OPTARG" ;;
116         o) OutputDirectory="$OPTARG" ;;
117         h) usage ;;
118         l) more $0; exit 1 ;;
119        \?) echo invalid argument, type $NAME_ -h for help; exit 1 ;;
120     esac
121
122 done
123 shift $(( $OPTIND - 1 ))
124
125 # args check
```

```
126 [[ $TemplateInputFile ]] || { echo >&2 No template input was specified; exit 1; }
127 [[ $SeedsFile ]] || { echo >&2 No SeedsFile specified.  Creating Random Seeds;
        CreateSeeds=1;}
128 [[ -d "$OutputDirectory" ]] || { echo >&2 output dir "$output_dir" does not exist;
        exit 1; }
129
130 declare -a nodes
131 declare -i CPUS_PER_NODE
132 declare -i MAX_CPUS
133 declare -a SeedsOfLife
134
135 GLOBAL_COUNTER=1
136 CPUS_PER_NODE=4
137 index=1
138 nodes=(`grep -v ^# /home/adan/hydrahosts|cut -c 1-7`)
139
140 let MAX_CPUS=${#nodes[*]}*4
141 echo "Max Number of CPUS is "$MAX_CPUS
142 #Seeds of Life
143 #declare -a SeedsOfLife
144 ###################################Seeds Of Life#####################################
145 ####IF SeedsFile was not specified, then an array of seeds will be created
146 if [[ $CreateSeeds ]]
147   then
148     for (( element=1; element<=$MAX_CPUS; element++))
149     do
150       let value=2*$RANDOM$RANDOM+1
151       #echo $value
152       SeedsOfLife[$element]=$value
153     done
154   else
155       SeedsOfLife=( $( < $SeedsFile ) )
156       #Put a Check here to see that it's at least as big as MAX_CPUS
157       NumberOfSeeds=(`cat $SeedsFile|wc -l`)
158       if [ "$NumberOfSeeds" -lt "$MAX_CPUS" ]
159         then
160           echo "You Do not have enough seeds in $SeedsFile";
161           echo "You have $NumberOfSeeds seeds and you need $MAX_CPUS seeds"
162           exit 1;
163       fi
164 fi
165 ###################################Seeds Of Life#####################################
166 for node in ${nodes[*]}
167 do
168   echo "Running on $node"
169    #echo $index
170    index2=1
171    while (("$index2" <= "$CPUS_PER_NODE"))
172    do
173      ssh $USER@$node "if [ ! -d $OutputDirectory/$GLOBAL_COUNTER ];
174                       then mkdir $OutputDirectory/$GLOBAL_COUNTER;
175                       fi;"
```

```
176       ssh $USER@$node cp $TemplateInputFile $OutputDirectory/$GLOBAL_COUNTER/input
177       echo Using seed = ${SeedsOfLife[$GLOBAL_COUNTER]} for $GLOBAL_COUNTER
178       ssh $USER@$node "sed -i 's/c rand seed RNSEED/dbcn 7j ${SeedsOfLife[
              $GLOBAL_COUNTER]}/g' \
179                        $OutputDirectory/$GLOBAL_COUNTER/input"
180       ssh -n -f $USER@$node "
181                        export DATAPATH=/usr/local/udata/mcnpxs;
182                        export TMPDIR=/tmp
183                        export INPUT_FILE=input;
184                        export OUTPUT_FILE=output;
185                        export RUN_TAPE=runtp;
186                        export LOG_FILE=log.txt;
187                        export STD_OUTERR=$HOME/local/$GLOBAL_COUNTER/stdouterr.txt;
188                        cd /$OutputDirectory/$GLOBAL_COUNTER;
189                        nohup mcnp-cp inp=\$INPUT_FILE out=\$OUTPUT_FILE runtpe=\
                               $RUN_TAPE \
190                        dumn1=\$LOG_FILE 2>&1>\$STD_OUTERR;
191                       "
192       let "GLOBAL_COUNTER++"
193       let "index2++"
194     done
195     let "index++"
196 done
```

B.10 BASH Script used to collect data output from various MCNP-CP Jobs

```
1  #!/bin/sh
2  declare -a nodes
3  declare -i CPUS_PER_NODE
4  declare -i MAX_CPUS
5  declare -a SeedsOfLife
6
7  GLOBAL_COUNTER=1
8  CPUS_PER_NODE=4
9  index=1
10 nodes=(`grep -v ^# /etc/hydrahosts|cut -c 1-7`)
11
12 let MAX_CPUS=${#nodes[*]}*4
13
14 if [ ! -d collected ]
15    then mkdir collected
16 fi
17
18 for node in ${nodes[*]}
19 do
20   echo "Collecting from $node"
21    index2=1
22    while (("$index2" <= "$CPUS_PER_NODE"))
```

```
23    do
24      if [ ! -d collected/$GLOBAL_COUNTER ]
25         then mkdir collected/$GLOBAL_COUNTER
26      fi
27      scp $node:/local/$USER/$GLOBAL_COUNTER/* collected/$GLOBAL_COUNTER/
28
29
30      startlines=(`grep -n "cell (" collected/$GLOBAL_COUNTER/output|cut -d ':' -f1
            `)
31      endlines=(`grep -n "      total      [0-9]" collected/$GLOBAL_COUNTER/output|
            cut -d ':' -f1`)
32      for i in "${!startlines[@]}"
33      do
34        startline=`expr ${startlines[i]} + 2`
35        endline=`expr ${endlines[i]} - 1`
36        sed -e "$startline,$endline!d" collected/$GLOBAL_COUNTER/output >
              $GLOBAL_COUNTER.out.$i
37        cat collected/$GLOBAL_COUNTER/output|grep "      total      [0-9]"|cut -c
              -17 --complement >$GLOBAL_COUNTER.TOTALS
38      done
39
40
41
42      #echo $GLOBAL_COUNTER
43      let "GLOBAL_COUNTER++"
44      let "index2++"
45    done
46    let "index++"
47  done
```

Appendix C

Paralleling Code Manually on CEAR Cluster

C.1 How the File System Works on the CEAR Cluster

On the CEAR cluster each individual computing node has its own fixed disk. This is important because software that uses intensive disk I/O does not work well on a network file system that is on a centralized server. However each node also uses a shared file system that is distributed from a central host called NFSSERVER. A centralized file system that is distributed to all the computing nodes is convenient because all the user files for all the computing nodes will be synchronized. A second advantage inherited from this type of infrastructure is redundancy and smaller down time. The computing host NFSSERVER which houses the all of the user files is on RAID5 with six two terabyte hard drives. The six hard drives each have four partitions 1, 2, 3, and 4 of sizes 300MB, 1.8TB, 198 GB and 2GB respectively. One hundred megabytes was left unused at the end of the disk as slack space. This is because whenever a drive is exchanged the new drive will probably not have the exact same size as the old drive. The first partition on all six hard drives is configured as ID FD. These first partitions on all six drives are then used to make a device of type RAID1 (Mirror) called /dev/md0. This device /dev/md0 serves as the boot device for the NFSSERVER host. Because all the data is mirrored exactly across all six start partitions of each of the six hard drives, the total size is still 300 MB. The second partition on all of the six drives is combined in RAID 5 to form a device called /dev/md1. This is the device that is mounted as /home and is exported via nfs across all of the computing nodes. The size of this device is approximately 8.1 Terabytes as it is the result of the size of the partitions times the number of partitions grouped minus one. The third device created is /dev/md2. This is the result of combining the third partition (200GB) across all size hard drives in RAID5. This serves as the root of the local file system for the NFSSERVER. Its total size is about nine

hundred gigabytes. Finally the last partition on the first 3 and last 3 drives are setup in RAID5 to create two devices, /dev/md3 and /dev/md4 which serve as virtual memory swap files for the NFSSERVER.

C.2 File System Layout of the Computing Nodes

Each computing node contains a 160 gigabyte hard drive that is partitioned in three. The first partition is the virtual memory swap file and is about 4 GB. The second partition is the root (/) of the local file system for the individual computing node. Its size is about 64 GB. Finally the third partition is allocated the remainder of the drive which ends up being about 90 GB once it has been formatted. On each individual node on the root (/) file system there exists two directories name local and home. The local directory is really a mount point for the third partition of the hard drive on that node. It represents local storage that is only available on that computing node. Finally home is mount point for the remote file system /home which is hosted on the NFSSERVER. In this way when a user logs onto any computing node, this user will be presented with the same files. Additionally each user has a directory inside their home directory called local. This directory called local is a symbolic link to a directory with the same name as the users login name within the directory /local. To clarify things further, say a user with login id batman connects and logs on to the host node100. This user at the same time logs onto the host node140. The files in the directory /home/batman will be almost identical on the two hosts except for the folder /home/batman/local. On node100 /home/batman/local is a pointer to /local/batman of node100. Similarly on node140, /home/batman/local points to /local/batman of node140. With this setup as one can imagine, it is possible to take advantage of a distributed file system and a local file system. The obvious advantage of the local file system is putting less stress on the network connectivity of the cluster. Also access times for reading and writing of files are faster. One of the reasons MCNP runs very fast is because all of the cross section data is actually loaded from the computing node's local hard drive. Normally MCNP is ran via MPI on the CEAR cluster from the users home directory and there for there is a single output file in the users home directory. This illustrates effective use of both local and distributed file systems working together. There are however some codes that have not had any type of parallelizing done to them, neither OpenMP nor MPI.

C.3 Parallelizing stand alone Codes on the CEAR Cluster

Because most of the codes used on the CEAR cluster are Monte Carlo codes, parallelizing them makes sense. The following example of parallelizing the MCNP-CP code should serve a recipe for other code, CEARCPG comes to mind. Running a single instance of a code is a pretty

easy task, however running it multiple times on multiple computers is really tedious if done manually. Further if the user whishes to then analyze the data from 41 computers which ran 4 cases each, there will be 164 files in different places to collect and then try to merge back together.

C.4 Step 1 - use of command line arguments

Like MCNP, MCNP-CP uses command line arguments. It is this property that facilitates the use of the program from a batch process. In the case of MCNP-CP, a BASH shell script called runmcnp-cp was created. The script is self documenting, if called from the command line without arguments, it will display usage information. The basic idea is this, because of the way the CEAR cluster is configured; any one user can execute programs on any computing node from any other computing node. By simply executing ssh node103 ps aux from say node100, the user will get a process status of all running programs on node103. Using ssh without the need to login is possible because each users home/.ssh directory is exactly the same on each host. The files authorized_keys, id_rsa, id_rsa.pub, and known_hosts have all been created for the user. This step was actually necessary to implement the Hydra Process Manager for running MPI jobs on the CEAR cluster. The shell script runmcnp-cp then uses this ability to run jobs on other computing nodes. The program itself also uses command line arguments as follows: A -t parameter specifies an MCNP-CP template file. This file is basically an input deck for MCNP-CP with a comment on the last line c rand seed RNSEED. A -n parameter specifies the number of computing nodes the user would like to use. This parameter is optional and may be omitted so that the maximum number of operational nodes gets used. Another optional parameter -s specifies a seeds files. These might be useful in case the same work needs to be replicated. If the -s parameter is omitted the BASH script simply generates random odd numbers to be used as initial seeds for each instance of MCNP-CP that is ran. A -o parameter specifies the output. Usually to take advantage of each computing node's hard drive, the user will specify -o ~/local. Of course the tilde on BASH is interpreted as the user's home directory.

When the cases for this research were ran a command such as the following was executed: "runmcnp-cp -t 5cmcp2 -o ~/local" What happened next is that on each node, inside each of the present user's local folder, four subfolders with a numeric name were created, (1 per processor core).

A total of 156 folder were created because 2 of the 41 computing nodes were down for repairs but the script was robust enough to figure this out and compensate appropriately. In each one of these directories the template file was copied as the name "input" and the line inside the

Table C.1: Nodes and their directories used for local disk I/O

Computing Node	node100	node102	node103	...	node140
Node Directory	/local/adan/1	/local/adan/5	/local/adan/9	.	/local/adan/153
Node Directory	/local/adan/2	/local/adan/6	/local/adan/10	.	/local/adan/154
Node Directory	/local/adan/3	/local/adan/7	/local/adan/11	.	/local/adan/155
Node Directory	/local/adan/4	/local/adan/8	/local/adan/12		/local/adan/156

input text file that read "c rand seed RNSEED" was replaced with something much like the following "dbcn 7j 462157163". Of course the last number was a different random number for each of the input files. The program script then sets up all the appropriate ENVIROMENT variables and executes "nohup mcnp-cp inp=input out=output runtpe=runtp dumn1=logfile 2>&1>stdouterr.txt inside each one of the four folders created. This ran MCNP-CP 4 times on each active node in the background. When each jobs finished an output file was created in each numeric sub directory.

C.5 Step 2 (Collecting the Data)

In order to collect the data into a single unified folder for processing, a separate shell script was written called getmycpdata. The program creates a folder called collected and copies all of the numeric sub folders in each of the nodes to this folder. Additionally all of the output files are parsed and their f8 tallies extracted and written to the disk as separate files with just numeric data. These files are named using the convention number.out.tallynumber, where number represents the numeric folder from which it was created and tallynumber is an integer value representing the first, second, or third, etc. tally number to appear in the output file.

C.6 Step 3 Unifying the Data

In the case for this work, most of the output files contained two f8 tallies. In one run for example 312 separate numeric tallies files were created with the following names: 1.out.0, 1.out.1,2.out.0,2.out.1,,156.out.0,156.out.1. A C code was created called SpecAdder which would open all of the tallies and average the contributions and then save the result in another file. The relative error is also converted to standard error and then propagation of this error is carried through.

C.7 Utilities Created

Some tools were created because of how frequent a set number of operations are performed. The extraction of the f8 tally from an MCNP output is a prime example of this. Typically users opens up the output in a text editor, finds the appropriate region, selects and copies it out into a data analysis program. The program textract.sh available on each node as /usr/local/bin/f8extractor. It will create one text file per tally occurrence for a given MCNP output file.

Another shell script written was cleanmylocal which deletes all of the user files inside the users /home/<username>/local folder for each of the active nodes. For Sysadmin work the programs "makelocal <username>" and "makesshkey <username>" were created. The first one sets up the local folder for the giver username on all active nodes. The second program generates the unique ssh keys for the given username and sets up the appropriate structure inside the users .ssh folder. This allows the user to ssh amongst all of the nodes without need of a password once the user has logged onto at least one node with the proper credentials.

Appendix D

Compiling MCNP-CP

D.1 Building MCNP-CP on Linux

MCNP-CP was received as an executable and later as two patch files for the MCNP 4c source code sometime in early 2012. After all the appropriate paper work was taken care of the files were uploaded onto an ftp server that was setup for this purpose.

MCNP-CP was only ever meant for a Microsoft Windows environment. This specific version provided by Dr. Berlizov also had a feature which could write out list mode data. The executable version was tested using Microsoft Windows XP. Later an attempt was made to rebuild the code from the patches provided. The attempt proved successful using Microsoft Visual Studio 6 and Compaq Visual Fortran. In order to simulate more histories it made sense to build a Linux version for use with the CEAR cluster. Additionally the intention was send back the result of this work to Dr. Berlizov. Seeing how it was possible to compile the original MCNP 4c, the task on a single node was begun to compile the source code for MCNP-CP.

D.2 Compiling MCNP 4c on Slackware Linux

The MCNP 4c source code is distributed as two files, a C and a Fortran source code file. A separate utility prpr is included that pre processes and patches the source code from the patch files. Finally fsplit (a file splitting program) will split a patched file into multiple files. The files can then be compiled using a Fortran compiler and a C compiler. The first challenge was finding an fsplit program that worked. The OpenBSD version of fsplit.c version 1.15 was found to work when combined with a version of strlcpy also borrowed from OpenBSD. The C compiler used was GCC. The Fortran compiler used was Intel Fortran. Recently the code was recompiled and the exact versions of the compilers used were GCC version 4.7.1 and Intel Fortran 12.1.6 on a

Slackware 14.0 Linux environment.

D.3 Compiliing MCNP-CP on Slackware Linux

There were a lot of difficulties in trying to compile a Linux version of MCNP-CP. There were modifications made to the C and the Fortran patches. Usually when building MCNP the bulk of the code is in Fortran. It seems that the only use for C is to provide display capabilities via the use of X11. MCNP-CP is different in that there are more functions in C that are called from within the Fortran source. The C patch file to create MCNP-CP had a lot of modifications made to it. Every attempt in modifying the patch files were made such that the product could still be built on MS Windows and a Linux environment.

The C patch was changed as follows. There is an introduction of a struct called gamma. Every appearance of this name was changed to AGamma. This is because GCC has a definition already in place for a gamma function that uses the previous name. C's pound define directives were added such that if UNIX was defined then certain functions introduced would be called upon their lowercase name with an underscore appended (because of symbolic name mangling across languages). Additionally if UNIX is defined then pound if directives change the code such that the directory separator character is forward slash instead of a back slash and the refrence to the conio.h file is removed.

The Fortran patch was not modified directly. Instead a build BASH script was made to compile the source code. This script uses sed (the unix stream editor) to make modifications to the patches during compile time. For the both the Fortran and C patches the compilation definitions are changed slightly. These changes define things for the building environment. Additionally on the C patch file "#include <stdlib.h>" is appended after the definitions. The source code was also compiled using Microsoft Visual Studio 2010 and the Intel(R) Visual Fortran Composer XE 2011 Update 12 for Microsoft Visual Studio just to test code portability.

Appendix E

Input Decks

E.1 MCNP-CP Input Deck for ring of Detectors

```
1  Rexon GPS-200N 2x2 NaI(Tl) Detectors 15cm for MCNP-CP
2  c Cell Cards
3  1 0 (-1 +2 -5 +8):(-1 -4 +5)                IMP:P=1 U=9
       $Detector Can
4  2 0 -2 -5 +6                                IMP:P=1 U=9                    $Rubber
       Pad
5  3 0 (-2 +3 -7 +8):(-2 -6 +7)                IMP:P=1 U=9                    $Outside
        Reflector
6  4 0 -3 -7 +8                                IMP:P=1 U=9
       $Detector Crystal
7  101 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=51    IMP:P=1 U=0
       $Alumiium Detector Can
8  102 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=51    IMP:P=1 U=0                    $BF-1000
        Silicon Rubber Pad
9  103 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=51    IMP:P=1 U=0
       $Aluminium Oxide Outside Reflector
10 104 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=51    IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
       Detector Crystal
11 201 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=52    IMP:P=1 U=0
       $Aluminium Detector Can
12 202 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=52    IMP:P=1 U=0                    $BF-1000
        Silicon Rubber Pad
13 203 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=52    IMP:P=1 U=0
       $Aluminium Oxide Outside Reflector
14 204 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=52    IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
       Detector Crystal
15 301 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=53    IMP:P=1 U=0
       $Aluminium Detector Can
16 302 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=53    IMP:P=1 U=0                    $BF-1000
        Silicon Rubber Pad
17 303 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=53    IMP:P=1 U=0
       $Aluminium Oxide Outside Reflector
```

```
304 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=53   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
401 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=54   IMP:P=1 U=0
    $Aluminium Detector Can
402 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=54   IMP:P=1 U=0                      $BF-1000
     Silicon Rubber Pad
403 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=54   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
404 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=54   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
501 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=55   IMP:P=1 U=0
    $Aluminium Detector Can
502 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=55   IMP:P=1 U=0                      $BF-1000
     Silicon Rubber Pad
503 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=55   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
504 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=55   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
601 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=56   IMP:P=1 U=0
    $Aluminium Detector Can
602 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=56   IMP:P=1 U=0                      $BF-1000
     Silicon Rubber Pad
603 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=56   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
604 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=56   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
701 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=57   IMP:P=1 U=0
    $Aluminium Detector Can
702 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=57   IMP:P=1 U=0                      $BF-1000
     Silicon Rubber Pad
703 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=57   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
704 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=57   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
801 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=58   IMP:P=1 U=0
    $Aluminium Detector Can
802 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=58   IMP:P=1 U=0                      $BF-1000
     Silicon Rubber Pad
803 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=58   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
804 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=58   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
901 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=59   IMP:P=1 U=0
    $Aluminium Detector Can
902 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=59   IMP:P=1 U=0                      $BF-1000
     Silicon Rubber Pad
903 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=59   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
904 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=59   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
111 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=71   IMP:P=1 U=0
    $Alumiium Detector Can
```

```
112 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=71   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
113 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=71   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
114 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=71   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
211 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=72   IMP:P=1 U=0
    $Aluminium Detector Can
212 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=72   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
213 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=72   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
214 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=72   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
311 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=73   IMP:P=1 U=0
    $Aluminium Detector Can
312 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=73   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
313 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=73   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
314 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=73   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
411 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=74   IMP:P=1 U=0
    $Aluminium Detector Can
412 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=74   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
413 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=74   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
414 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=74   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
511 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=75   IMP:P=1 U=0
    $Aluminium Detector Can
512 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=75   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
513 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=75   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
514 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=75   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
611 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=76   IMP:P=1 U=0
    $Aluminium Detector Can
612 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=76   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
613 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=76   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
614 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=76   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
711 LIKE 1 BUT MAT=1 RHO=-2.7    TRCL=77   IMP:P=1 U=0
    $Aluminium Detector Can
712 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=77   IMP:P=1 U=0                        $BF-1000
     Silicon Rubber Pad
713 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=77   IMP:P=1 U=0
    $Aluminium Oxide Outside Reflector
```

```
714 LIKE 4 BUT MAT=4 RHO=-3.667  TRCL=77   IMP:P=1 U=0 LLD=1.15 ULD=1.45  $NaI
    Detector Crystal
1000 0 -1000 #101 #102 #103 #104                        &
             #201 #202 #203 #204                        &
             #301 #302 #303 #304                        &
             #401 #402 #403 #404                        &
             #501 #502 #503 #504                        &
             #601 #602 #603 #604                        &
             #701 #702 #703 #704                        &
             #801 #802 #803 #804                        &
             #901 #902 #903 #904                        &
             #111 #112 #113 #114                        &
             #211 #212 #213 #214                        &
             #311 #312 #313 #314                        &
             #411 #412 #413 #414                        &
             #511 #512 #513 #514                        &
             #611 #612 #613 #614                        &
             #711 #712 #713 #714      IMP:P=1 U=0  $Inside Universe
1001 0 +1000                          IMP:P=0 U=0  $Outside Universe

c SURFACE CARDS
1 CY +2.8448
2 CY +2.7940
3 CY +2.5400
4 PY 0.0000
5 PY -0.0508
6 PY -0.2032
7 PY -0.3048
8 PY -5.3848
1000 so 1000

c DATA CARDS
*TR51 0 -15 0 0 90 90 90 0 90 90 90 0
*TR52 0 -13.85819299 5.740251485 0 90 90 90 22.5 112.5 90 -67.5 22.5
*TR53 0 -10.60660172 10.60660172 0 90 90 90 45 135 90 -45 45
*TR54 0 -5.740251485 13.85819299 0 90 90 90 67.5 157.5 90 -22.5 67.5
*TR55 0 0 15 0 90 90 90 90 180 90 0 90
*TR56 0 5.740251485 13.85819299 0 90 90 90 112.5 202.5 90 22.5 112.5
*TR57 0 10.60660172 10.60660172 0 90 90 90 135 225 90 45 135
*TR58 0 13.85819299 5.740251485 0 90 90 90 157.5 247.5 90 67.5 157.5
*TR59 0 15 0 0 90 90 90 180 90 90 90 180
*TR71 0 13.85819299 -5.740251485 0 90 90 90 202.5 292.5 90 112.5 202.5
*TR72 0 10.60660172 -10.60660172 0 90 90 90 225 315 90 135 225
*TR73 0 5.740251485 -13.85819299 0 90 90 90 247.5 337.5 90 157.5 247.5
*TR74 0 2.75658E-15 -15 0 90 90 90 270 360 90 180 270
*TR75 0 -5.740251485 -13.85819299 0 90 90 90 292.5 22.5 90 202.5 292.5
*TR76 0 -10.60660172 -10.60660172 0 90 90 90 315 45 90 225 315
*TR77 0 -13.85819299 -5.740251485 0 90 90 90 337.5 67.5 90 247.5 337.5
m1 13027 1                            $Aluminum
m2 6000 .2 1000 .6 8000  .1 14000 .1 $BF-1000 Polydimethylsiloxane
m3 8000 .4 13000 .6                   $Aluminum Oxide
m4 11000 .5 53000 .5                  $NaI
```

```
Mode P
c Source
SDEF POS=0 0 0 ZAM=0270600
CPS 50 1 1 0 0 0 0 0 0 1 1
c Energy bins for response calculation
c Starting from 104
F108:P ((104) +{+(104) +(204)})       $22.5
F118:P ((104) +{+(104) +(304)})       $45
F128:P ((104) +{+(104) +(404)})       $67.5
F138:P ((104) +{+(104) +(504)})       $90
F148:P ((104) +{+(104) +(404)})       $112.5
F158:P ((104) +{+(104) +(604)})       $135
F168:P ((104) +{+(104) +(704)})       $157.5
F178:P ((104) +{+(104) +(804)})       $180
c Starting from 204
F208:P ((204) +{+(204) +(304)})       $22.5
F218:P ((204) +{+(204) +(404)})       $45
F228:P ((204) +{+(204) +(504)})       $67.5
F238:P ((204) +{+(204) +(604)})       $90
F248:P ((204) +{+(204) +(704)})       $112.5
F258:P ((204) +{+(204) +(804)})       $135
F268:P ((204) +{+(204) +(904)})       $157.5
F278:P ((204) +{+(204) +(114)})       $180
c Starting from 304
F308:P ((304) +{+(304) +(404)})       $22.5
F318:P ((304) +{+(304) +(504)})       $45
F328:P ((304) +{+(304) +(604)})       $67.5
F338:P ((304) +{+(304) +(704)})       $90
F348:P ((304) +{+(304) +(804)})       $112.5
F358:P ((304) +{+(304) +(904)})       $135
F368:P ((304) +{+(304) +(114)})       $157.5
F378:P ((304) +{+(304) +(214)})       $180
c Starting from 404
F408:P ((404) +{+(404) +(504)})       $22.5
F418:P ((404) +{+(404) +(604)})       $45
F428:P ((404) +{+(404) +(704)})       $67.5
F438:P ((404) +{+(404) +(804)})       $90
F448:P ((404) +{+(404) +(904)})       $112.5
F458:P ((404) +{+(404) +(114)})       $135
F468:P ((404) +{+(404) +(214)})       $157.5
F478:P ((404) +{+(404) +(314)})       $180
c Starting from 504
F508:P ((504) +{+(504) +(604)})       $22.5
F518:P ((504) +{+(504) +(704)})       $45
F528:P ((504) +{+(504) +(804)})       $67.5
F538:P ((504) +{+(504) +(904)})       $90
F548:P ((504) +{+(504) +(114)})       $112.5
F558:P ((504) +{+(504) +(214)})       $135
F568:P ((504) +{+(504) +(314)})       $157.5
F578:P ((504) +{+(504) +(414)})       $180
c Starting from 604
F608:P ((604) +{+(504) +(704)})       $22.5
```

```
F618:P ((604) +{+(504) +(804)})        $45
F628:P ((604) +{+(504) +(904)})        $67.5
F638:P ((604) +{+(504) +(114)})        $90
F648:P ((604) +{+(504) +(214)})        $112.5
F658:P ((604) +{+(504) +(314)})        $135
F668:P ((604) +{+(504) +(414)})        $157.5
F678:P ((604) +{+(504) +(514)})        $180
c Starting from 704
F708:P ((704) +{+(704) +(804)})        $22.5
F718:P ((704) +{+(704) +(904)})        $45
F728:P ((704) +{+(704) +(114)})        $67.5
F738:P ((704) +{+(704) +(214)})        $90
F748:P ((704) +{+(704) +(314)})        $112.5
F758:P ((704) +{+(704) +(414)})        $135
F768:P ((704) +{+(704) +(514)})        $157.5
F778:P ((704) +{+(704) +(614)})        $180
c Starting from 804
F808:P ((804) +{+(804) +(904)})        $22.5
F818:P ((804) +{+(804) +(114)})        $45
F828:P ((804) +{+(804) +(214)})        $67.5
F838:P ((804) +{+(804) +(314)})        $90
F848:P ((804) +{+(804) +(414)})        $112.5
F858:P ((804) +{+(804) +(514)})        $135
F868:P ((804) +{+(804) +(614)})        $157.5
F878:P ((804) +{+(804) +(714)})        $180
E0 0 1023I 3.0
NPS 100000000
c rand seed RNSEED
```

E.2 MCNP Input Deck used for cross talk of various angles

```
Rexon GPS-200N 2x2 NaI(Tl) Detectors 5cm for MCNP
c Cell Cards
1 0 (-1 +2 -5 +8):(-1 -4 +5)                IMP:P=1 U=9  $Detector Can
2 0 -2 -5 +6                                IMP:P=1 U=9  $Rubber Pad
3 0 (-2 +3 -7 +8):(-2 -6 +7)                IMP:P=1 U=9  $Outside Reflector
4 0 -3 -7 +8                                IMP:P=1 U=9  $Detector Crystal
101 LIKE 1 BUT MAT=1 RHO=-2.7     TRCL=51   IMP:P=1 U=0  $Aluminium Detector Can
102 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=51    IMP:P=1 U=0  $BF-1000 Silicon Rubber
    Pad
103 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=51    IMP:P=1 U=0  $Aluminium Oxide Outside
    Reflector
104 LIKE 4 BUT MAT=4 RHO=-3.667   TRCL=51   IMP:P=1 U=0  $NaI Detector Crystal
201 LIKE 1 BUT MAT=1 RHO=-2.7     TRCL=58   IMP:P=1 U=0  $Aluminium Detector Can
202 LIKE 2 BUT MAT=2 RHO=-0.1922 TRCL=58    IMP:P=1 U=0  $BF-1000 Silicon Rubber
    Pad
203 LIKE 3 BUT MAT=3 RHO=-3.9700 TRCL=58    IMP:P=1 U=0  $Aluminium Oxide Outside
    Reflector
204 LIKE 4 BUT MAT=4 RHO=-3.667   TRCL=58   IMP:P=1 U=0  $NaI Detector Crystal
1000 0 -1000 #101 #102 #103 #104                         &
```

```
16                #201 #202 #203 #204            IMP:P=1 U=0  $Inside Universe
17  1001 0 +1000                                 IMP:P=0 U=0  $Outside Universe
18
19  c SURFACE CARDS
20  1 CY +2.8448
21  2 CY +2.7940
22  3 CY +2.5400
23  4 PY 0.0000
24  5 PY -0.0508
25  6 PY -0.2032
26  7 PY -0.3048
27  8 PY -5.3848
28  1000 so 1000
29
30  c DATA CARDS
31  *TR51 0 -5 0 0 90 90 90 0 90 90 90 0 $Initial Detector on the Left
32  *TR54 0 -1.913417162 4.619397663 0 90 90 90 67.5 157.5 90 -22.5 67.5 $67.5
33  *TR55 0 0 5 0 90 90 90 90 180 90 0 90 $90
34  *TR56 0 1.913417162 4.619397663 0 90 90 90 112.5 202.5 90 22.5 112.5 $112.5
35  *TR57 0 3.535533906 3.535533906 0 90 90 90 135 225 90 45 135 $135
36  *TR58 0 4.619397663 1.913417162 0 90 90 90 157.5 247.5 90 67.5 157.5 $157.5
37  *TR59 0  5   0 0 90 90 90 180 270 90 90 180 $180
38  m1 13027 1                                   $Aluminum
39  m2 6000 .2 1000 .6 8000  .1 14000 .1 $BF-1000 Polydimethylsiloxane
40  m3 8000 .4 13000 .6                          $Aluminum Oxide
41  m4 11000 .5 53000 .5                         $NaI
42  Mode P
43  c Source
44  sdef  par=2 pos=0 0 0  erg=D1 vec=0 -1 0 dir=D2
45  si1  L 1.1732 1.3325 0.3471 0.8261 2.1586 2.5057
46  sp1  D 0.499635189 0.500289156 3.77295E-05 3.2838E-05 5.07463E-06 1.25089E-08
47  si2  -1 0.869166 1
48  sp2  0 0.934583 0.065417
49  SB2  0. 0. 1.
50  c Energy bins for response calculation
51  F8:P 104 204
52  F18:P 104 204
53  FT8 GEB 0.011131 0.036071 1.4056
54  F1:P 201004
55  c E18 0 1023I 3.0
56  E0 0 1023I 3.0
57  E1 0 1023I 3.0
58  NPS 1000000000
```