Investigating the Use of the Intel Xeon Phi for Event Reconstruction

Keegan H. Sherman and G.P. Gilfoyle
Physics Department, University of Richmond

Jefferson Lab

The goal of Jefferson Lab (JLab) is to understand how quarks and gluons combine to form nucleons and nuclei. The laboratory is undergoing an upgrade that will double the beam energy to 12 GeV and build a new detector in Hall B, CLAS12. CLAS12 will collect data at a prodigious rate. Here we describe our investigation of new hardware (the Intel Xeon Phi) to speed up reconstruction, simulation, and analysis of the data.

The CLAS12 Detector

- CLAS12 (Figure 1) is a large acceptance spectrometer that takes data over a large solid angle.
- Data will be collected for eight detector subsystems for the base equipment with more than 60,000 channels.
- CLAS12 will have an event rate of 10 kHz and each event is 10 kB. It will take 5-10 TB of data per day. We expect to need 12,000 cores to keep up with the reconstruction, simulation, and analysis of the data [1].
- In this work we are studying a new technology to speed the processing of the data.

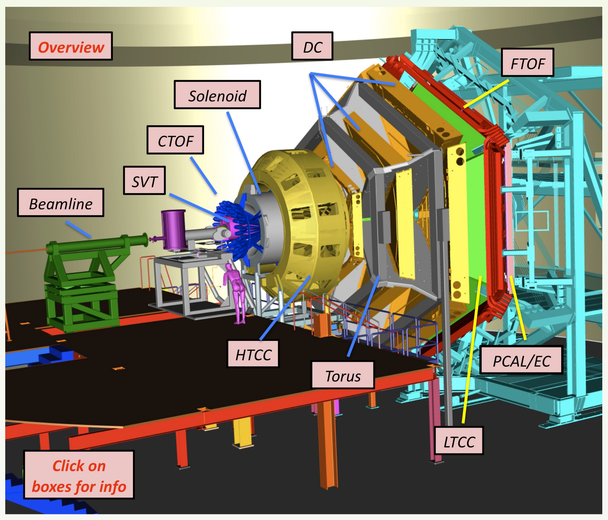


Figure 1. Computer designed image of CLAS12.

Intel Xeon Phi

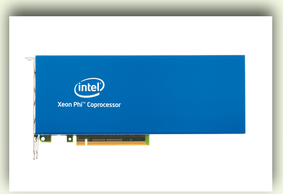
We are exploring the use of the Xeon Phi, shown in Figure 2. The Phi is a chip designed to run highly parallel computations quickly. Its specifications are shown in the Table 1 [2].

Table 1. Hardware\software specifications for the Xeon Phi.

Number of Cores	Threads per Core	Register Width	Number of Registers	Number of Mask Registers	Compatible Languages
60	4	512 bits	32	8	C, C++, Fortran

The Phi has its own unique instruction set called the Many Integrated Cores (MIC) architecture which requires Intel's compiler Composer XE.

Figure 2. The Intel Xeon Phi.



The Kalman Filter

One of the most CPU-time-consuming steps in the data reconstruction is the Kalman Filter used to determine the particle trajectory from the drift chamber data (see Figure 1). It is a linear algebra algorithm designed to extract signals from noisy data [3]. It is summarized below.

- Take an initial point x_{n-1} (a five dimensional state vector) and covariance matrix P_{n-1} and use the state transition matrix A, control matrix B, control vector u_n, and process error covariance matrix Q to predict the next point x_{np} and covariance matrix P_{np}.

$$x_{np} = Ax_{n-1} + Bu_n$$
$$P_{np} = AP_{n-1}A^T + Q$$

- Calculate the comparison values $\tilde{y}$ and S using the predicted values with the actual values z_n and R and observation matrix H.

$$\tilde{y} = z_n - Hx_{np}$$
$$S = HP_{np}H^T + R$$

- Next compute the Kalman Gain K.

$$K = P_{np}H^T S^{-1}$$

- Use the Kalman Gain to correct predictions.

$$x_{ne} = x_{np} + K\tilde{y}$$
$$P_{ne} = (I - KH)P_{np}$$

- Use the results to go to the next data point.

Writing the Filter for the Phi

We have written a simplified Kalman Filter algorithm in C++ to test the Phi using a five-dimensional state vector. Each file has been optimized to make full use of the extra wide registers the Phi provides as well as the threads it can support.

DVector.h - Code to store multiple pieces of data in the ultra wide registers but also access each individual element if needed.

Matrix2x2.h/.cpp
Matrix2x3.h/.cpp
Matrix3x2.h/.cpp
Matrix3x3.h/.cpp
Matrix5x1.h/.cpp
Matrix5x5.h/.cpp

Matrix classes to define matrix addition, subtraction, multiplication, and transposition. The square matrices also define how to find the determinant and the inverse.

KalmanFilter.h/.cpp - This code defines the actual steps of the Kalman Filter. It takes in matrices and data and then runs the filter.

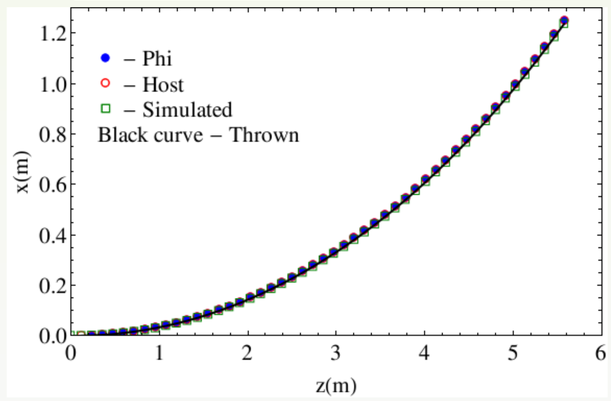
Testing the Phi

We tested the Kalman Filter in two ways, accuracy of the results, and the time it took to run on the Phi vs. the host machine. We simulated the motion of a charged particle in a magnetic field at 4 GeV, 7 GeV, and 10 GeV.

Accuracy Test

For this test we ran the 4 GeV set of data through the filter once on both the Phi and the host to check that both were calculating reasonable answers. Figure 4 shows the input function, simulated data, and reconstructed results from the Phi and the host.

Figure 4. Path of an electron in a constant magnetic field.



Time Test

We tested the algorithm with 500 events for ten separate runs at each energy. Table 2 shows how we configured the Phi and the host for each test. For the host, we ran the algorithm in *Mathematica*. Table 3 shows our results in time per 500 events. Overall the Phi ran much faster. The uncertainties of our data come from overhead of running multiple threads and background processes on the processors.

Table 2. Configuration of the Xeon Phi and Xeon host processor for testing.

	Cores used	Threads per Core	RAM size
Xeon Phi	60	3	8 GB
Host	12	1	64 GB

Table 3. Results of the time test between the Phi and the host.

Time per 500 events	4 GeV	7 Gev	10 GeV
Xeon Phi	0.66 ± 0.24 sec.	0.78 ± 0.25 sec.	0.75 ± 0.26 sec.
Host	342.71 ± 2.31 sec.	343.41 ± 1.32 sec.	342.77 ± 1.96 sec.

Conclusion

We have compared a Kalman Filter algorithm on the Xeon Phi coprocessor and its host CPU. The Xeon Phi was significantly faster. We also tested the accuracy of our filter and found that the host and Phi results largely agree.

Future Work

We are working on implementing the algorithm in C++ on the host. We also observe some differences between the Kalman filter reconstruction and the thrown data that we will investigate. Finally, we are studying other ways to compare the Phi with other processes.

References

[1] D.P. Weygand and V. Ziegler, editors, 'CLAS12 Software', Jefferson Lab, 2012. http://clasweb.jlab.org/clas12/software/clas12_software_tdr_2012.05.pdf

[2] Intel (2014). Intel Newsroom http://newsroom.intel.com/docs/DOC-3126

[3] Czerniak, G. (2014). Kalman Filter http://greg.czerniak.info/guides/kalman1/