PARALLELISM IN PYPWA

C. Salgado (NSU/JLab), M.Vick (Governor's School),
M. Jones(NSU), W. Phelps (FIU)

Overview

- Introduction to Amplitude Analysis
- Motivation for Parallel Computing in Amplitude Analysis
- Multiprocessing in Python
- Introduction to the Xeon Phi
- Amplitude Analysis on the Xeon Phi
- Future directions of MIC architecture
- Summary

Introduction to Amplitude Analysis

- Amplitude Analysis is an umbrella term used to describe analyses that use amplitudes to describe the physics of decays/reactions
- Example: 3-body decay of $\omega/\phi \to 3\pi$
- These amplitudes will be used in a **maximum likelihood fit** and they take a set of event dependent kinematic variables and overall fit parameters
- PyPWA is the framework that is focused on ease of use and simplification of the process of Amplitude Analysis

$$I(\vec{x}, \vec{a}) \equiv \sum_{ext.\ spins} |\mathscr{M}|^2$$

Expected Number of Events

$$-ln\mathscr{L} = -\sum_{i=1}^{N} ln\left[I(\vec{x}, \vec{a})\right] + \mathscr{N}.$$

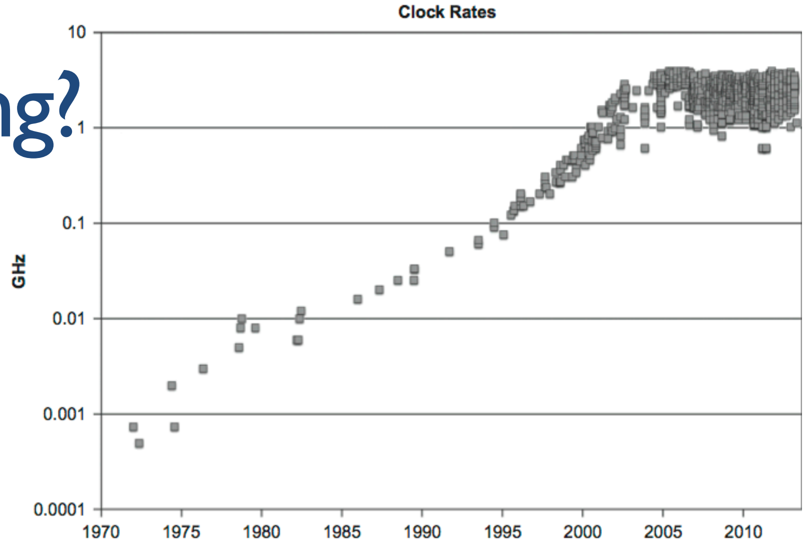
Sum over data

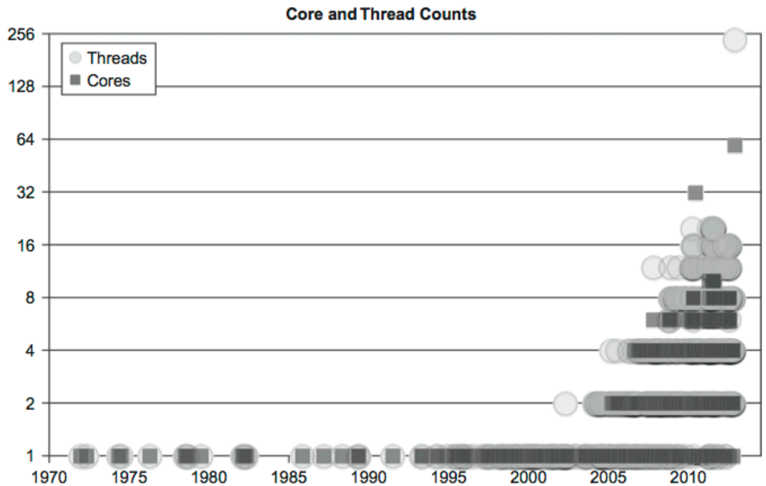
Kinematic variables

Fit parameters

Why Parallel Computing?

- Fitting in Amplitude Analysis is a computationally intensive task
  - It is not uncommon that the calculation of the likelihood function would take several hours due to the complexity of the theoretical amplitudes
- The trend over the past decade has been an increase in the number of cores/threads per CPU
  - On an average workstation this allows for a significant speedup
- To keep up with an increasing amount of data and stagnant serial performance, we must use parallel computing in the future

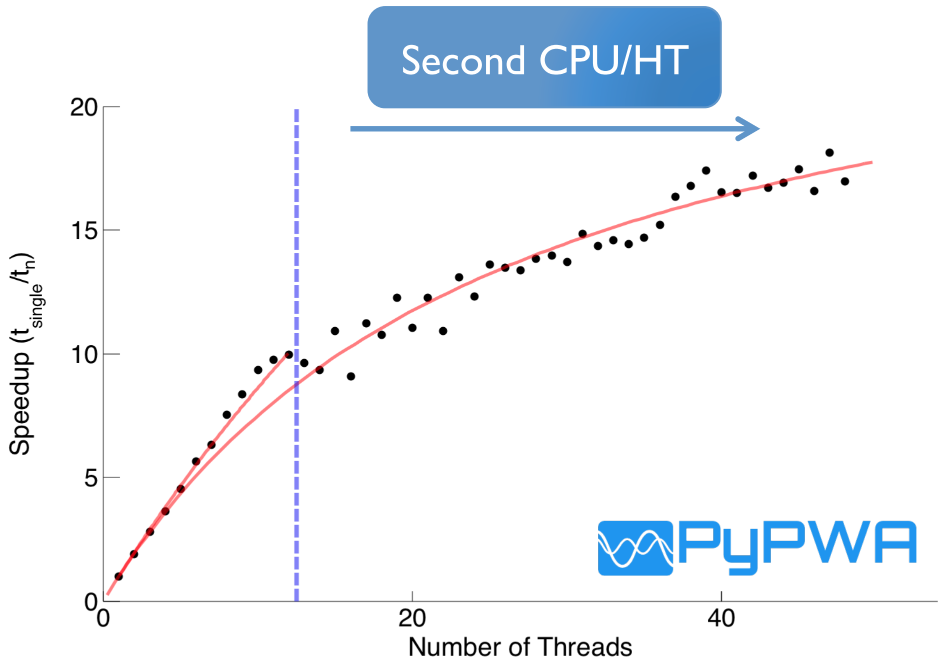




Figures: Jim Jeffers and James Reinders

Python Multiprocessing

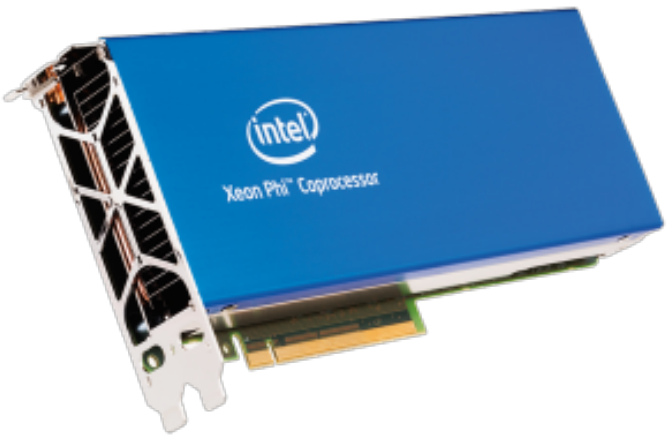
- The first attempt to speed up the fit process involves some modification to the PyPWA framework
- Python multiprocessing works well for splitting up the likelihood function
  - Multiprocessing was used to skirt around the GIL which prevents more than one python interpreter running per process
- Provided excellent speedup compared to the serial version and is now included in the production build



Model	#Processors	Cores/CPU	Threads/Core	Total #Threads
Xeon E5-2670v2	2	12	2	48

Xeon Phi Development (Knight's Corner)

- Many Integrated Cores (MIC) Technology
  - ~60 cores x 4 threads/core = ~240 threads
- Little to no code modification to get up and running but modifications needed later to tune the code for higher performance
- Theory Amplitudes written in Fortran compile with **no modification**!
  - The FORTRAN code would likely need **rewriting** if used with any **GPGPU** accelerator card
- After compiling comes integration with C/C++ likelihood function and testing
- Many possible ways to utilize the card: Intel's TBB, pthreads, Cilk, and **OpenMP**



OpenMP on the Xeon Phi (or CPU)

- OpenMP is a library developed to allow for easy to use portable shared memory multiprocessing and vectorization
- This library functions using #pragmas which contain work sharing constructs and clauses
- Allows for parallelization and vectorization without altering the meaning of the original code
  - Program will still compile serially if the compiler does not support OpenMP!

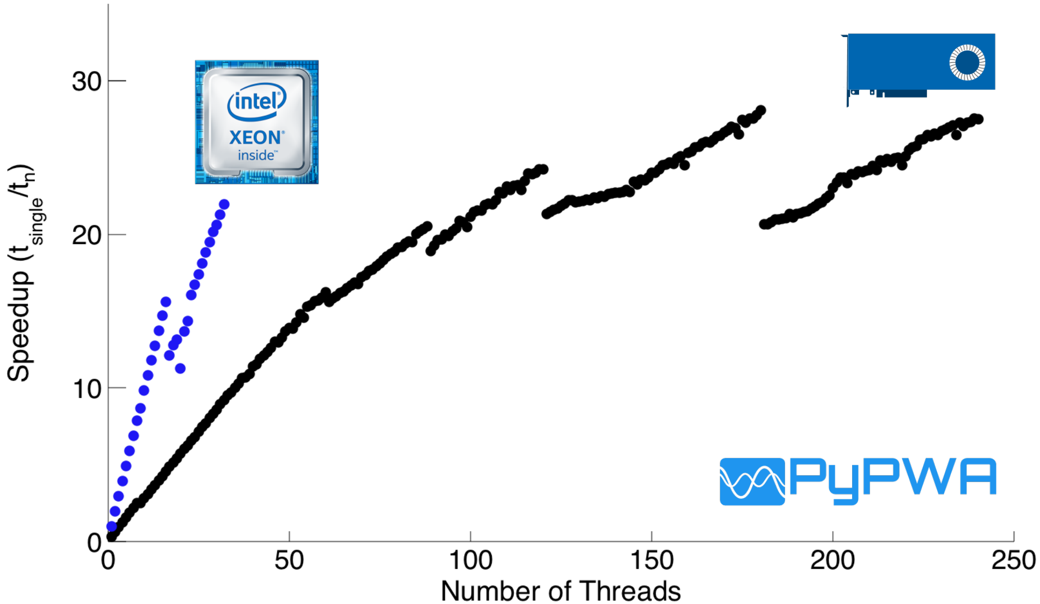
```
omp_set_num_threads(numthreads);
tstart = dtime();
#pragma omp parallel{
    #pragma omp for
    for(x=0;x<events;x++){
        A[x]=igor_amplitude(res,S[x],T[x],U[x],p);
    }
    lnl(A,events);
}
tstop = dtime();
```

Spawns worker threads (`#pragma omp parallel{`)

Parallelizes for loop (`#pragma omp for`)

MIC and CPU Scaling

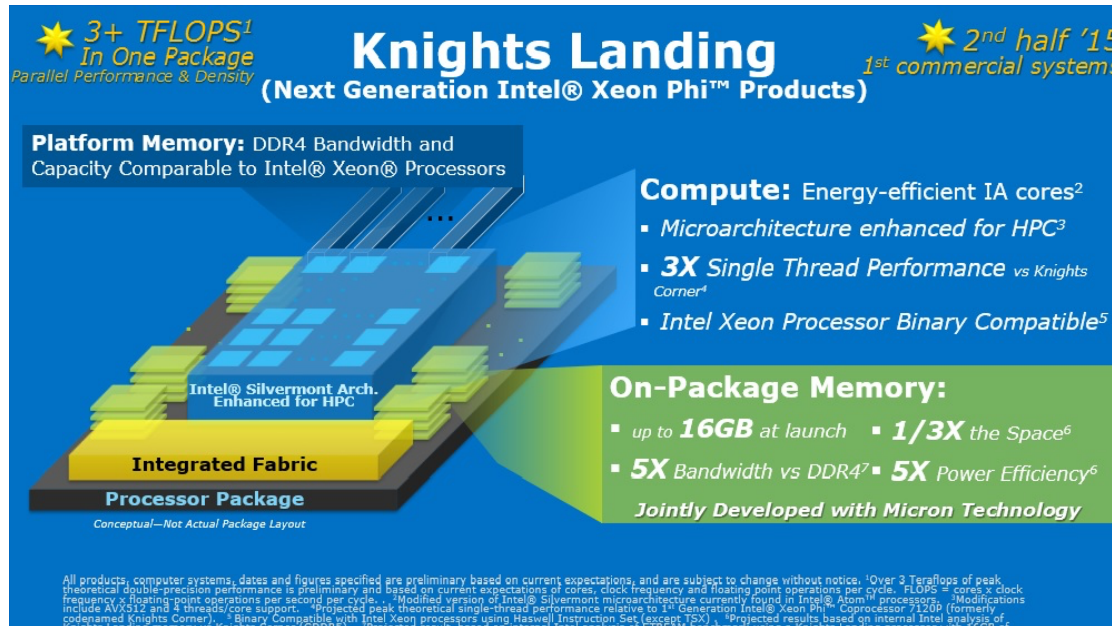
- Xeon Phi performance normalized to CPU performance
- The results from the first successful test are encouraging for using the Xeon Phi for production analyses
- Xeon Phi code is only multiprocessed here, with vectorization the performance could be **8x** higher!



Model	#Processors	Cores/CPU	Threads/Core	Total # Threads
Xeon E5-2650	2	8	2	32
Xeon Phi 5100P	1	60	4	240

Future directions of MIC Architecture

- Knight’s Landing:
  - 72 Silvermont cores
  - Shipping in 2016!
  - Socketed and PCI-Express versions available
  - **Back to homogeneous computing?**



Summary

- Multiprocessing has been utilized to increase performance in Amplitude Analysis using Python Multiprocessing and OpenMP on the Xeon Phi
- A performance increase of ~23x on a CPU and ~30x on the Xeon Phi has been seen using OpenMP
- Performance optimizations and further benchmarks will be pursued before integration into the production build
- For more information:
  - http://pypwa.jlab.org
  - https://github.com/JeffersonLab/PyPWA