Second Interim Report

Yanqi Gu

Mentor: Sharon Brunett

Topic: cWB Optimization for GW Signal Waveform Reconstruction Stage

1.Pipeline Profiling

More timing:

I tried different methods of timing, set timing triggers inside the target functions to see the time cost of different parts of that function. Different timing methods are used and compared to find the best way to count the really short time duration.

Deeper background understanding:

I extract the math equations of the whole processing, separate them into different parts and relate them to corresponding parts in the code. Optimizing opportunities are in reorganization of likelihood Wavelet parameters, the data extraction from sky locations by traversing the whole sky.

Experiments on different machines with different compilers:

The runtime performance of a pipeline relies largely on hardware and compiling way. In our experiment, we use three compilers: GCC, ICC, NVCC to build the pipeline. We put most of our work on two of Caltech's computing nodes, one with Intel Xeon(32 cores) and NVIDIA Tesla P100, another with Xeon Phi(256 cores). We also use a dedicated node to avoid competing usage of resources with others.

Time consuming:

Below are simple time-consuming results for different machines and different compilers:

ICC on Xeon Phi(TM) CPU 7210 1.30GHz using one CPU: 4:47:31

GCC on Xeon Phi(TM) CPU 7210 1.30GHz Xeon Phi using one CPU: 4:34:06

ICC on Xeon(R) CPU E5-2670 2.60GHz using one CPU: 1:21:03

GCC+NVCC on Xeon(R) CPU E5-2670 2.60GHz using one CPU: 1:20:12

ICC on Xeon(R) CPU E5-2630 v3 2.40GHz using one CPU: 1:01:41

2.Analysis of cWB pipeline

Steps:

Initialization-> Read Data ->Data Conditioning ->Coherence Analysis ->Supercluster processing ->Likelihood Calculation ->Data Save

Above are basic steps of the whole pipeline, the time-consuming parts is the last step, which includes the reconstruction step.

Loops:

Loop structure is our first target to parallelize, in cWB pipeline there're several parameters that are basic elements of loop in computing:

Sky location: The coordinates in the whole sky. The sky is divided into many locations (in this case the number is 196608), each location is expressed by an integer. These locations are projected to a sky map and traversed by HEALPix.

Pixel: The data collected by detectors are pixels in the sky.

Cluster: A cluster includes lots of pixels.

Lag: Data packages. Each lag has several clusters.

(p.s. The value of cluster, pixels and lags are determined by different datasets. In our dataset there're 6576 clusters,153321 pixels and 590 lags)

After analysis, we decide to parallelize the likelihood Wavelet Packet loop because it traverse all lags and sky locations in two loops.

Method to parallelize:

One potential method is using manycore processor with parallel language. We are trying to run open MP code on Xeon and Xeon Phi using ICC and GCC compiler.

Another method is to use GPU and CUDA language for acceleration. However, one problem is that most part of likelihoodWP loop is written using NETX structure and SSE, which is Intended for 8 threads running cWB using CPU. To use CUDA, first we need to translate the SSE version back to simple floats, and then translate that to CUDA. It would be a large project, but first we choose to recode the DPF calculation (cost 25% overall time) back to float version to see the performance.

3.Open MP implementation

As we target the loop in likelihoodWP, we seek for parallel methods to reduce the computational complexity. However, problems happen when implementing Open MP.

Problem:

1. The computing resources are shared so it's hard to make the most use of the computing capacity. However, this problem should be solved by 1. choosing an idle time to run the pipeline 2. using GPU 3. building a dedicated node
2. Sometimes the process just run away with some unknown reason.
3. Change of input parameters: During our investigation period, cWB authors made updates to key parameters which influenced the duration of benchmarking runtime, increasing the time needed for profiling and timing runs of interest. Recently, we have discovered some useful shortcuts to help profiling just portions of interest without running the full benchmark (taking more than 1 hr.).
4. Change of algorithm: The primary time-consuming function _avx_norm_ps we thought we'd be optimizing is no longer the time-consuming function due to algorithmic optimization by the author. Now we changed the target to function _avx_dpf_ps which is used to calculate DPF patterns.

4. CUDA application

Right now, we have built the pipeline with NVCC and tested the runtime performance. We have designed the testing structure for recoding specified function with CUDA. We are now recoding the DPF function, which is a time-consuming function cost 25% runtime.

5.Research goals

We continue with our goal to optimize the cWB pipeline with parallel methods, reconstruction of data and running on various architectures with acceleration promise.