LArSoft vectorization plan

Status and plans for studies of LArSoft vector acceleration for proto(DUNE) on x86 platforms

Agathangelos Stylianidis (Angelos)(UoP), Nektarios Benekos (CERN), Andrea Dell'Acqua (CERN)

Introduction - Presentation Purposes

- Present our plans on how to improve LArSoft performance using vectorization.
- Work management planning and establishment of the means of our collaboration.
- At the end I would like to hear your comments about our approach and **I will also tell you what we need from LArSoft developers.**

Why vectorization is important?

Why vectorization is important?

- Single thread performance is the base of optimizations.
- Speedup through vectorization is predictable.
- Can reach the limits of CPUs computation capabilities.

CPUs support

Vectorization parallelism exists in every CPU core

- Streaming SIMD Extensions (SSE)
- Fused Multiply - Add (FMA)
- Advanced Vector Extensions (AVX)

Methodology

1. Profiling to locate the time consuming parts of the program.
2. Prediction studies of the performance gain using vectorization.
3. Apply vectorization.
4. Testing
   - 4.1 Verification: Make sure the code is still correct.
   - 4.2 Performance evaluation: Gain or loss. We gain speed-up cross check with our predictions. In case of loss, find out what when wrong.

Methodology - Profiling

- gprof
- perf
- gdb

Methodology - Apply vectorization

We need vectorized code that can be run on x86 architecture, CPU agnostic

We will try to trigger the compiler to write the vector instructions for us

Methodology - Apply vectorization

1. Compile with compiler optimizations enabled
2. Get a report from the compiler what optimizations were applied, e.g. which part of the code was vectorized and which part of the code was not vectorized due to other dependencies
3. Examine the not vectorized code due to dependencies and try to help the compiler to resolve the dependencies

Future Steps - Other single thread performance optimizations

Future Steps - Other single thread performance optimizations

- Exploit data locality
  - spatial locality
  - temporal locality
- Profile the program memory accesses
- Cache-oblivious algorithms (if possible)
- Code guideline optimizations

Code guideline optimizations

1. Follow AMDs guide Software Optimization Guide for AMD64 Processors
2. More C++ specific guidelines and optimizations.

Cache-oblivious algorithms

”A cache-oblivious algorithm is designed to perform well, <u>without modification</u>, on multiple machines with different cache sizes, or for a memory hierarchy with different levels of cache having different sizes.”
- Wikipedia

What has been done so far

- Download and compiled LArSoft v07.06.00
- Test some of the protoDune_xx.fcl job options
- However there are some open questions

Open questions

- How to speed-up the compilation if we apply changes to one file.
- protoDune.root test files (evgen, g4, detsim, reco) are not coming with the releases. Am I wright?
- How to verify that I have not broken the software.

Your Comments