openlab/intel hackathon-workshop

Paul Seyfert

CERN

18th May 2017

workshop homepage

we participated in three teams with fluctuating members

- global overall performance picture
- thread safety
- analyse one algorithm in detail

workshop homepage

we participated in three teams with fluctuating members

- global overall performance picture
  Intel(R) VTune(TM) Amplifier
- thread safety
  Intel(R) Inspector
- analyse one algorithm in detail
  Intel(R) Advisor

overall

- a large amount of time spent for setting up
  - recompile the entire software stack starting from gaudi
  - realise debug symbols are needed (recompile again)
  - realise tools are bound by disk i/o (afs)
  - realise you don't want to run a gui through x-forwarding
  - realise memory diagnostics needs an additional kernel module

amplifier: lessons learned

- initialising MiniBrunel (e.g. loading detector geometry) takes long
  ⇒ added `IntelProfiler` branch to `lhcb/Gaudi`
- we spend a lot of time in memory allocation
- PrPixel had monitoring enabled in the future branch
  ⇒ speedup for free! (but not wrt. the last round of performance numbers)

advisor: lessons learned

- output with gcc much less impressive than what we were shown with Intel's compiler
  - less information about why auto vectorisation failed
  - (code that's vectorised by hand often not visible as such)

Ineffective masked remainder for AVX512 codes

- Compiler generates vector masked remainder due to the number of iterations (trip count) not being divisible by vector length. In case of executing a few iterations, it is ineffective comparing to scalar versions of the loop.
- Using AVX512 mask profiler and trip-counts data to prove the issue.

Recommendation: Force scalar remainder generation Confidence: Low

The compiler generated a masked vectorized remainder loop that contains too few iterations for efficient vector processing. A scalar loop may be more beneficial. To fix: Force scalar remainder generation using a directive: `#pragma simd novecremainder` or `#pragma vector novecremainder`.

Example: Force the compiler to not vectorize the remainder loop

```
void add_floats(float *a, float *b, float *c, float *d, float *e, int n)
{
    int i;
    #pragma simd novecremainder
    for (i=0; i<n; i++)
    {
        a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
    }
}
```

```
#pragma simd reduction(+:mean)
for(int j = 0; j < size; j++) {
        mean += data[order[j]] / N;
        data[order[j]] = 10.f / (j+1);
}
```

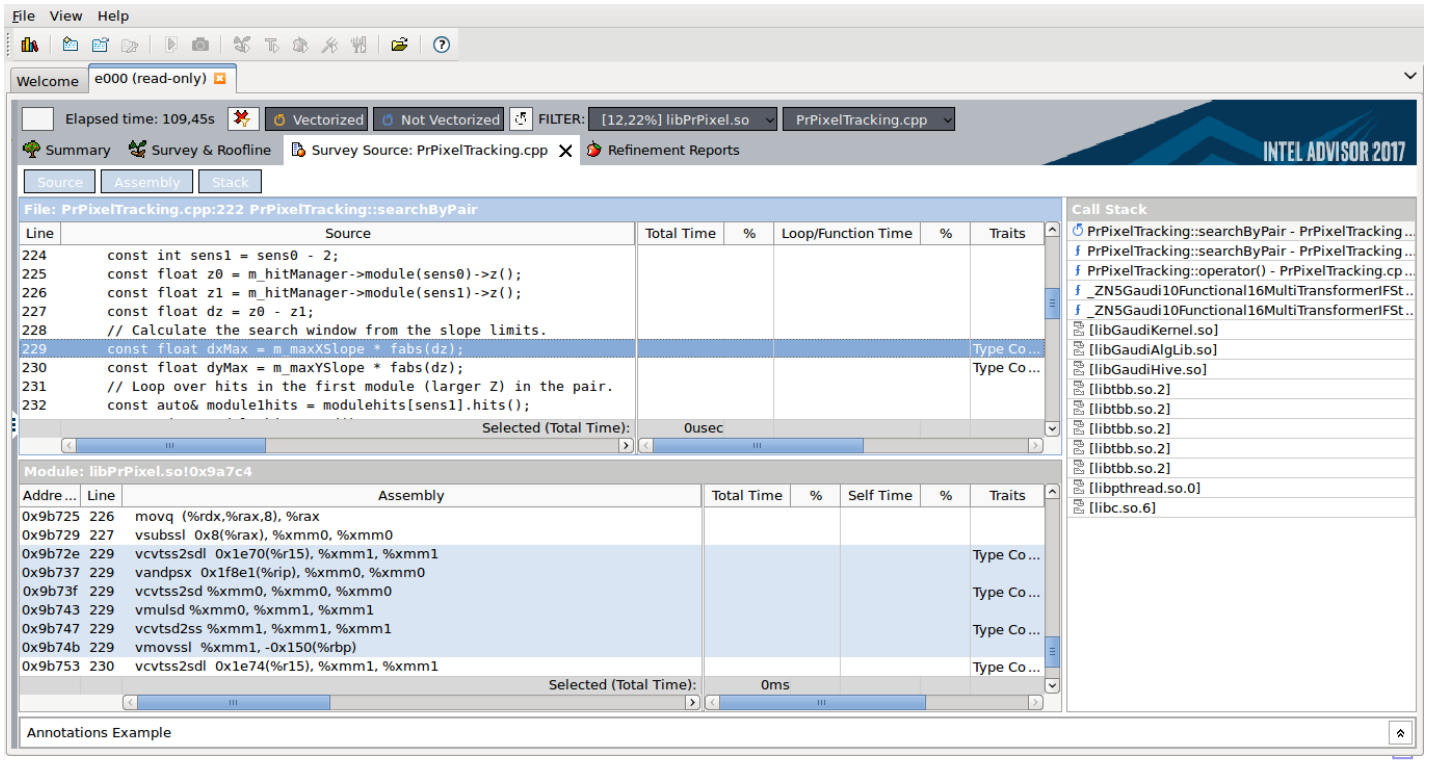
E.g. bad performance if ((size) % (loop_body_vl) == 1), in case of float number it results in 12.5% mask bits utilization only, in addition leads to gathers, scatters...

Read More:

- simd, vector
- Getting Started with Intel Compiler Pragmas and Directives and Vectorization Resources for Intel® Advisor Users

advisor: lessons learned

- output with gcc much less impressive than what we were shown with Intel's compiler
  - less information about why auto vectorisation failed
  - (code that's vectorised by hand often not visible as such)
  - notifications about `float`→`double`→`float` conversions that disappear in the `opt` build anyway

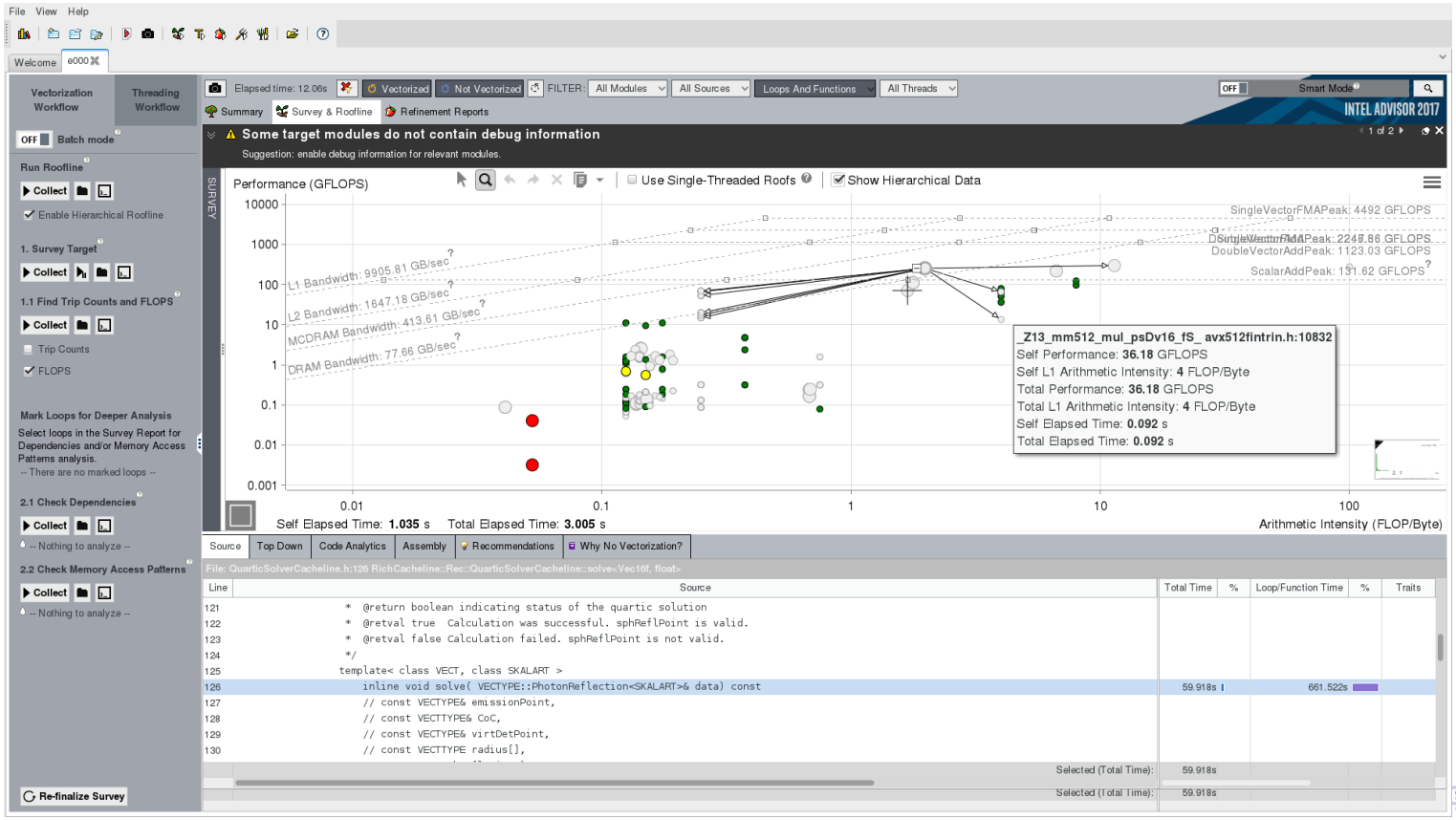


advisor: lessons learned II

- tool developed for fully parallelised runs
  → reference lines far off at first
  → having the experts there to adjust the settings a big plus
- post processing of profiles unfeasible on afs
- most analyses require two runs
  (10 min with the latest advisor on openlab machine vs. hours on online machine with the slightly older version)

advisor output: roofline

- y-axis: how much math we do (higher = better)
- x-axis: how much math per data we do
  - left: CPU could do more math, but memory will not be able to give enough data
  - right: CPU will be the bottle neck



situation in our code

- mostly in the middle where it's hard to tell if we'll hit memory or CPU limits first
- often well away from the performance limit of the machine
- grouping of functions problematic

Summary

a bunch of features not explored

- snapshots: pack source and assembly and profile into archives (to be able to manipulate code and still have valid references, have it portable to laptop to investigate)
- recommendations and examples for design patterns

organisatorial

- having experts at hand helps getting running!
- expect that building with icc brings a better experience working with the intel tools
- having the profiler and the profile in hands alone doesn't magically speed things up