Intel profiling tools and roofline model

Dr. Luigi Iapichino
luigi.iapichino@lrz.de
Which tool do I use in my project? A roadmap to optimization (and to the next hour)

We will focus on tools developed by Intel, available to users of the LRZ systems.

Again, we will skip the MPI layer.

VTune is a very rich tool, we will touch it only quickly.

We will dedicate more time to Advisor.

But first we introduce the roofline model, a crucial tool in code modernization.
Profiling with Intel® VTune Amplifier XE

- Powerful tool for analyzing the node-level performance
  - Multiple programming languages (C/C++, Fortran, .NET, Java, Assembly)
  - Support for all latest Intel® processors (incl. Intel® MIC / Broadwell micro-architectures)
- Performance analysis at different levels
  - High-level (code analysis, parallelization efficiency), no special rights needed
  - Low-level (inspection of all architectural components), module driver is required
  - Processor-specific analysis (e.g., utilisation of vector units on Intel® MIC)
- Minimal execution time overhead
  - No recompilation or special linking needed
  - H/W counter sampling and multiplexing → all interesting events gathered once
- Multiplatform (Windows/Linux, 32/64-bit) + complete command-line interface
- Can produce very large traces (~400MB per min. of exec. time)
Hot-spot guided optimization

Typical workflow

1. Compile code with -g -O2 or -g -O3
2. Set the environment variables or use a wrapper script
3. Tweak code input for a short representative run
Performance overview

Wall-clock time
Cumulative CPU time
Performance bottlenecks are highlighted in red
Overall CPU usage

Elapsed Time: 15.338s
CPU Time:
- Effective Time:
  - Idle: 0s
  - Poor: 29.472s
  - Ok: 48.818s
  - Ideal: 12.906s
  - Over: 0s
Spin Time:
- Imbalance or Serial Spinning (OpenMP): 9.698s
- Lock Contention (OpenMP): 50.802s
Overhead Time:
- Total Thread Count: 16
- Paused Time: 0s

CPU Usage Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

Intel Vtune Amplifier XE
Threads behaviour

- Function level profiling
- Time line of the application
Threads behaviour: locks and waits

- Threads are spinning!
- Threads sleeping
- Useful work
- Concurrency
- Synchronization

Intel Vtune Amplifier XE
Source code view

```
88  /* Scale display coordinates to actual region */
89  c.real = real.min + ((double) col * scale.real);
90  c.imag = imag.min + ((double) (height-1-row) * scale.imag);
91  /* height-1-row so y axis displays */
92  * with larger values at top */
93  /*
94  /* Calculate z0, z1, ..., until divergence or maximum iterations */
95  int k = 0;
96  double lengthsq, temp;
97  do {
98      temp = z.real*z.real - z.imag*z.imag + c.real;
99      z.imag = z.real*z.imag + c.imag;
100     z.real = temp;
101     lengthsq = z.real*z.real + z.imag*z.imag;
102     +k;
103   } while (lengthsq < (MN) & k < maxiter);
104  /* Scale color and save */
105  colors[col] = (long) ((h-1) * scale_color) + min_color;
106  /* Display row */
107  /*
108  *pragma omp critical
109  {
110     for (int col = 0; col < width; ++col) {
111        if (setup_return == EXIT_SUCCESS) {
112         XSetForeground (display, gc, colors[col]);
113         XDrawPoint (display, win, gc, col, row);
114        }
115     }
116  }
117  free (colors);
118  ```
Closing remarks

The tool is useful and can be used to find:

- Hotspots in the code and possible bottlenecks
- Characterization of the parallelization efficiency
- Possible locks and spinning threads in the application

- More advanced profiling is provided using special kernel modules (memory bandwidth, hardware event-based sampling, ...)

- Instrumenting the code for reducing the amount of profiling part in the application
Roofline model
Roofline model

The roofline model allows to understand the performance limit of an application, based on operational intensity (algorithm specific) and on hardware specifics (memory bandwidth).

The expected performance is defined as:

\[ P = \min(P_{\text{max}}, I b_s) \]

- \( P_{\text{max}} \): Applicable peak performance of a loop, assuming that data comes from L1 cache (this is not necessary \( P_{\text{peak}} \)).
- \( I \): Computational/arithmetic intensity ("work" per Byte transferred) over the slowest data path utilized ("the bottleneck").
- \( b_s \): Applicable peak bandwidth of the slowest data path utilized.
Roofline model

Peak performance of 2-socket Ivy-Bridge node

Peak: 448 GFlops/s
Roofline model

Peak performance of 2-socket Ivy-Bridge node

Peak: 448 GFlops/s
Stream BW: 78.5 GB/s
Roofline model

Peak performance of 2-socket Ivy-Bridge node

Peak: 448 GFlops/s
Stream BW: 78.5 GB/s
Roofline model

Peak performance of 2-socket Ivy-Bridge node

Peak: 448 GFlops/s
Stream BW: 78.5 GB/s
Core Performance: 22.4 GFlops/s
The core parameter behind the Roofline model is Arithmetic Intensity. Arithmetic Intensity is the ratio of total floating-point operations to total data movement (bytes). A BLAS-1 vector-vector increment (\(x[i] += y[i]\)) would have a very low arithmetic intensity of 0.0417 (N FLOPS / 24N Bytes) and would be independent of the vector size.
Intel® Advisor XE
Profiling with Intel® Advisor XE

- Modern HPC processors explore different levels of parallelism: between the cores (multi-threading), within a core (vectorization).

- Adapting applications to take advantage of such high parallelism is often defined as code modernization.

- Intel® Advisor XE is a software tool for vectorization optimization and thread prototyping.

- The tool guides the software developer to resolve issues during the vectorization process.
Creating a new project via GUI

Interface similar to VTune
Setting up the application

Command-line parameters

Environment variables
Vectorization analysis workflow

1. Run Survey
2. Check the Trip-counts
3. Check Dependencies
4. Check Memory Access Patterns

Edit & compile
Take Snapshot
Deeper-dive analysis
(Mark-up Loops)
5 Steps to efficient vectorization

1. Compiler diagnostics + Performance Data + SIMD efficiency information

2. Guidance: detect problem and recommend how to fix it

3. “Accurate” Trip Counts + FLOPS: understand utilization, parallelism granularity & overheads

4. Loop-Carried Dependency Analysis

5. Memory Access Patterns Analysis

Intel Advisor XE
Profiling with Intel® Advisor XE

- How to improve performance
- ISA
- Hot-spots
- What prevents vectorization
- Report from the loop
Profiling with Intel® Advisor XE

Vectorization informations
Number of vector registers
Traits
Loop features
### Profiling with Intel® Advisor XE

#### Higher instruction set architecture (ISA) available

Your application was compiled using an ISA lower than the ISA available on this machine. Consider recompiling your application on this machine using the highest ISA available – use the xHost option, or on the original machine for a higher ISA – use the x option.

#### Recommendations to enable vectorization

- **Issue:** Data type conversions present
  - There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

- **Recommendation:** Use the smallest data type
  - Use the smallest data type that gives the needed precision to use the entire vector register width.
  - **Example:** If only 16-bits are needed, using a short rather than an int can make the difference between eight-way or four-way SIMD parallelism, respectively.

---

**Useful suggestion**
Loop vectorized (base)

Vector length

Vectorization efficiency

Loop analytics
Loop vectorized (ver4) and improved gain
Memory access pattern (ver4)

Stride distribution
Roofline analysis with Advisor - base

Performance (GFLOPS)

Source Trip Counts: 2000

Traits

Square Roots, Type Conversions, Unpacks

GFLOPS: 1.02436

Static Instruction Mix

Memory: 15
Compute: 32
Mixed: 1
Other: 15
Number of Vector Registers: 15

Code Optimizations

Compiler: Intel(R) C++ Compiler for applications running on Intel(R) 64
Version: 17.3.4.196 Build 20170411
Roofline analysis with Advisor - ver4

The outputs of Advisor used for this session are under /lrz/sys/courses/KNL/advisor-analysis
Please try yourself!
Closing remarks: 6 steps vectorization methodology

1. **Measure baseline release build performance:**
   define a metric which makes sense for the code

2. **Determine hotspots using Intel® VTune:**
   most-time consuming functions in the application

3. **Determine loop candidates using compiler report:**
   -qopt-report=5 -qopt-report-phase=loop,vec

4. **Get advise using Intel® Advisor:**
   use the vectorization analysis capability of the tool

5. **Implement vectorization recommendations**