Performance Analysis Tools

1. Profiler Generalities

part of the material presented is courtesy of Fabio Baruffa, Intel and Luigi Iapichino, LRZ
Moore’s law is dead! Long live Moore’s law!

“There ain’t no such thing as a free lunch”

42 Years of Microprocessor Trend Data

- # of components still grows 😊
- Single-thread performance and frequency settle 😐
- Machines are more difficult to use
- Pressure for code optimization

Optimization flowchart & Intel® Profiling Tools

Since 2020:
Profiler
VTune™ Amplifier

This morning:
VT Ad

This afternoon:
(Anupam)
APS

Image courtesy of Intel®
### How profilers operate

<table>
<thead>
<tr>
<th>Statistical sampling</th>
<th>Event-based / Instrumenting / Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What they do</strong></td>
<td>Modify the executable, typically at <em>compilation time</em>, to call performance counters at any occurrence of specified event.</td>
</tr>
<tr>
<td>Interrupt the execution of the program to take <em>snapshots</em> of the process status: functions, call tree, memory, … May run the application several times <em>under the hood.</em></td>
<td></td>
</tr>
<tr>
<td><strong>Pros and cons</strong></td>
<td></td>
</tr>
<tr>
<td>✓ <em>Typically</em> no need to recompile</td>
<td>✗ Need to compile a dedicated version</td>
</tr>
<tr>
<td>✓ <em>Typically</em> negligible overhead</td>
<td>✗ Large overhead (e.g. small functions called many times)</td>
</tr>
<tr>
<td>✓ Runtime meaningful</td>
<td>✗ Runtime (and partials) <em>not indicative</em> of true performance</td>
</tr>
<tr>
<td>✗ <em>Statistical</em> nature of sampling: measure uncertainties, may need to tune sampling interval</td>
<td>✗ Complete performance characterization</td>
</tr>
<tr>
<td><strong>When preferred</strong></td>
<td></td>
</tr>
<tr>
<td>• First overview / blind profiling</td>
<td>• Development, debugging</td>
</tr>
<tr>
<td>• Profiling of production runs</td>
<td>• When time granularity &lt; few ms</td>
</tr>
<tr>
<td>• All the other cases</td>
<td>• Isolated kernels in controlled tests</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td></td>
</tr>
<tr>
<td>APS, VTune, ITAC, Advisor, perf (linux).</td>
<td>Scalasca, Score-P, …</td>
</tr>
</tbody>
</table>

### Two options on Intel; also how VTune actually works

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector (aka <em>event-based sampling</em>, watch out!uju)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses OS interrupts</td>
<td>Uses the on chip Performance Monitoring Unit (PMU)</td>
</tr>
<tr>
<td>Collects from a single process tree</td>
<td>Collect system wide or from a single process tree.</td>
</tr>
<tr>
<td>~10ms default resolution</td>
<td>~1ms default resolution (finer granularity - finds small functions)</td>
</tr>
<tr>
<td>Either an Intel® or a compatible processor</td>
<td>Requires a genuine Intel® processor for collection</td>
</tr>
<tr>
<td>Call stacks show calling sequence</td>
<td>Optionally collect call stacks</td>
</tr>
<tr>
<td>Works in virtual environments</td>
<td>Works in a VM only when supported by the VM (e.g., vSphere*, KVM)</td>
</tr>
<tr>
<td>No driver required</td>
<td>Uses Intel driver or perf if driver not installed</td>
</tr>
</tbody>
</table>

**Mixed strategies also possible**
(e.g. VTune some time ago)


**LRZ:**
- sep available on Linux Cluster
- perf available on SuperMUC-NG
  - `/proc/sys/kernel/perf_event_paranoid` is set to 0 on compute nodes (uncore events)
Performance Analysis Tools

2. Intel® VTune™ Profiler

part of the material presented is courtesy of Fabio Baruffa, Intel and Luigi Iapichino, LRZ
What VTune can do for you

- Analyze all aspect of node-level performance
  - threading communication SIMD memory access
- Identify bottlenecks, hints how to address them, predicts speedup
- List hotspots by execution time or resources consumed
- Optimize use of hardware resources (SIMD, hyperthreading, additional memories, …)
- Allow low-level hardware inspection (microarchitecture, frontend, backend)
- Group and display collected data by:
  - Process Thread Function Source code line …
- Point to target location in source code and shows assembly instructions
- Support multiple programming languages
  - (C/C++/C#, Fortran, Java, Assembly, Python, Google Go)
- Support latest and fortcoming Intel® processors
- Provide a flexible workflow integrating GUI and command line
What you can do for VTune

1. Get it!
   - Your own: standalone or in Intel Parallel Studio or Intel oneAPI  
     https://software.intel.com/en-us/advisor
   - LRZ: module load amplifier_xe [/2020]
     amplxe-gui / amplxe-cl

2. Install drivers for hardware collection (optional, need admin rights!)

3. Compile and run your code with profiler-friendly features
   - Compile with -g -O2 (or -O3) options, plus SIMD, e.g. -xHost (see lecture on Advisor)
   - Hyperthreading and Turbo Boost technology are discouraged
   - Keep it simple! Ideally a short, representative run on a single node.

4. Get familiar with the workflow and include it in your own

   1. Start VTune Profiler
   2. Configure and Run Analysis
   3. Analyze Performance Data

   Optimize!
VTune workflow 1/2

VTune Collecting samples

module load amplifier_xe

GUI `amplxe_gui` then fill the boxes. Also via SSH! CLI best on batch servers, multi-node.

CLI `amplxe_cl -help collect [analysis]`

```
[mpirun -n <N> / srun] amplxe_cl \ 
  -collect <analysis_1> -knob <name_1a>=<value_1a> -knob <name_1b>=<value_1b> ... \ 
  -collect <analysis_2> -knob <name_2a>=<value_2a> -knob <name_2b>=<value_2b> ... \ 
  -collect ... -r <dir> -- <program>
```

<table>
<thead>
<tr>
<th>Common analyses</th>
<th>Most useful corresponding knobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>** sampling_interval= &lt;msec&gt; [def: 5]**</td>
<td>* sampling-mode=hw [def: sw], analyze-openmp=true [def: false], enable-stack-collection=true, collect-memory-bandwidth=true</td>
</tr>
<tr>
<td>hpc-performance</td>
<td>enable-stack-collection=true</td>
</tr>
<tr>
<td>threading</td>
<td>* sampling-and-waits=hw [def: sw]</td>
</tr>
<tr>
<td>hotspots</td>
<td>* uarch-exploration</td>
</tr>
<tr>
<td>threading</td>
<td>* advanced-hotspots</td>
</tr>
<tr>
<td>* uarch-exploration</td>
<td>analyze-openmp=true, enable-characterization-insights=true</td>
</tr>
</tbody>
</table>

* Drivers required. Intel Sampling Driver (sep, check with sep -version) or, if not available, perf counters.
** Depending on finalization mode, data in excess of 500 MB may be skipped. A coarser time sampling is advised in case. One can always tweak -finalization-mode=full | fast | deferred | none. Check with amplxe_cl -help collect
Generating reports

module load amplifier_xe

good idea whenever a graphic environment is available

Alternatively, to compare two

CLI

amplxe-cl -help report [report_name]
amplxe-cl -report <report_name> -r <directory>

summary
hotspots
hw-events
callstacks
top-down

...
Navigating VTune: Reports and Views

Reports
- Somewhat dependent by collection

Views
- Available view depends on above
- but very similar across them
- Interface full of help / hints
VTune Views

Summary

Bottom-up

Calls

Source/Assembly
HPC Performance Characterization

How to browse
- Click and drag on a region to zoom

Summary-like report, or Call Stack
- Red flags = bottlenecks

VTune Reports

Performance Tools | 2020-03-13

12
VTune Reports

Hotspots / Threading efficiency

Same information, but you can focus on single threads
VTune Reports

Microarchitecture Exploration

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load.

This metric represents how much Core non-memory issues were of a bottleneck. Shortage in hardware

This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency ratio" (Actual Instructions Retired/Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe

Performance Tools | 2020-03-13
VTune Example #1

Examining Threading efficiency

- Useful for OpenMP applications.
- Where are my threading inefficiencies?
- What’s my potential gain?
VTune Example #2
Examining Memory Usage

- On which memory are my data?
- How are they moved around?
- What's the available bandwidth?
- Cache usage: how many misses?
Performance Analysis Tools

3. Vectorization recap

part of the material presented is courtesy of Fabio Baruffa, Intel and Luigi Iapichino, LRZ
Intel Advanced Vector instruction sets, AVX

Recap on SIMD (Single Instruction, Multiple Data)

An additional level of loop parallelism

Intel®Advanced Vector Extensions (Intel®AVX)

- Also AVX512 on 512-bits \( \text{zmm} \) registers (KNL™, Xeon™ Scalable)
- Naive vectorization of loops/kernels enabled at compile time with the following flags:
  - \(-xHost\)
  - \(-xMIC\text{-AVX512}\)
  - \(-xCORE\text{-AVX512} \ [-qopt-zmm\text{-usage}=high]\)
  - \(-no\text{-vec} -no\text{-simd}\)

Most advanced set on compiling machine

Forces AVX512 on KNL

Intel Xeon version (SKX, CKX) [default: no \( zmm \)]

Forces scalar code (e.g. for performance evaluation)

Node-level performance optimization

The possibilities of vector code

"Automatic" Vectorization Not Enough
Explicit pragmas and optimization often required

Threading and vector synergise

Much faster than separately

The difference grows with every new hardware generation!

Different levels of optimization:

- Compiler flags
  - Automatic, in seconds
-Pragma directives
  - "No" code change
-Intrinsic functions
  - Scalar won't compile
-Assembly
  - Good for inspection

More human-friendly → More efficient
SIMD Facts 1/2

Just another degree of parallelism

- It is limited by own Amdahl’s law:

\[
T_N = T_{\text{Serial}} + \frac{T_{\text{Parallel}}}{N} \rightarrow T_{\text{SIMD}} = T_{\text{Scalar}} + \frac{T_{\text{Vector}}}{\text{Length}_{\text{SIMD}}}
\]

- May be prevented by own “race conditions“:

**Loop-carried dependencies**

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAR</td>
<td>Write after read (anti dependency)</td>
<td>A[i] = A[i+1] + C[i]</td>
</tr>
<tr>
<td>WAW</td>
<td>Write after write (output dependency)</td>
<td>A[i%2] = B[i] + C[i]</td>
</tr>
</tbody>
</table>

• OpenMP has got you covered! 😊

From OpenMP 5.0 ref guide:
SIMD Facts 2/2

Must optimize memory resources!

- Need to work on many data at the same time to reach performance peak!
- Also must bring data to processor as fast as possible → optimize cache use!

Best use of memory for vector loops?
- **Alignment**: compilers generate scalar peel and remainder loop.
- **Data layout**: stride of memory access unit > constant > irregular

- *Jumps* with new SIMD instruction sets
- Often main parallel bottleneck
- Even less chances for free lunches!
Optimising Vectorization Performance with Intel® Advisor™
The Roofline model (in one slide)

Answers the question: up to which point can I optimize? Theoretical limits?

- Advisor generates them automatically
- Publication quality, tweaking required
- Cache-aware, and ungroup callstacks
- Nearest roofs point to most relevant bottlenecks
Performance Analysis Tools
4. Intel® Advisor™

part of the material presented is courtesy of Fabio Baruffa, Intel and Luigi Iapichino, LRZ
Optimising Vectorization Performance with Intel® Advisor™

What Advisor does, and how to get started

- A software tool for vectorization optimization and thread prototyping
- Focus on node-level code modernization
- Makes the most of Intel Processors

(Threading workflow &) Vectorization workflow

Roofline Analysis
Optimize your application for memory and compute.

Vectorization Optimization
Enable more vector parallelism and improve its efficiency.

Thread Prototyping
Model, tune, and test multiple threading designs.

Build Heterogeneous Algorithms
Create and analyze data flow and dependency computation graphs.

Get Advisor
https://software.intel.com/en-us/advisor

On LRZ machines
module load advisor_xe[/2020]

Get started
Advisor Vectorization Workflow 1/2

Collection

0. [module load advisor_xe] advixe-cl / advixe-gui advixe-cl -help [action]

1. [mpirun -n <N>] advixe-cl -collect <analysis> [--opt1 --opt2 ...] \ --project-dir=<path> -- <program>

Analyses
- Survey: Discover where efficient vectorization and/or threading pays off the most.
- Tripcounts: Identify the number of loop iterations, collects FLOPs
- Roofline: Shortcut for Survey + Trip Counts & FLOP, produces roofline plot

Useful options you may want (check help)
- --trace-mpi
- --enable-cache-simulation
- --stacks

2.0 [mpirun -n <N>] advixe-cl --mark-up-loops --project-dir=<path> --loops="scalar,has-source,has-issues,..." -- <program>

2.1 [mpirun -n <N>] advixe-cl -collect dependencies [--opt1 --opt2 ...] \ --project-dir=<path> -- <program>

Identify and explore loop-carried dependencies preventing vectorization for marked loops.

2.2 [mpirun -n <N>] advixe-cl -collect map [--opt1 --opt2 ...] \ --project-dir=<path> -- <program>

Identify and explore complex memory accesses reducing vector efficiency for marked loops.
module load advisor_xe

good idea whenever a graphic environment is available

GUI advixe-gui

CLI advixe-cl -help report [report_name]
advice-cl -report <report_name> --proj-dir=<directory>

summary hotspots survey dependencies roofline map ...

Generating reports Advisor Vectorization Workflow 2/2
Summary view

Look at:
- red flags
- recommendations

Warning: Timers are actual data, efficiencies are model results!

Toggle either on/off
Survey view: Code Analytics Tab

Bottom view: details on what you select
Navigating Advisor reports 3/4

Survey view: Recommendation Tab

Bottom view: details on what you select
Roofline view

Larger, more time-consuming kernels make better optimization candidates

Bottom view: details on what you select
### Problems preventing vectorization

#### Loop calls non-SIMD or non-inline functions
- Function vectorizable? Declare as SIMD
  - OpenMP**: `#pragma omp declare simd`  
- Only some loops vectorizable?  
  - May force vectorization on those: `#pragma omp simd`  

#### Loop-carried dependencies suspected*
1. Run dependencies analysis  
2. Clear? Force vectorization! No? Rewrite your code, e.g. changing loop schedule or data layout.

#### Inner loop already vectorized
- Enough iteration in inner loop? Never mind!  
- Not enough? Collapse *nested* loops:  
  - `#pragma omp simd collapse(n)`

#### Reduction clause needed (+, *, max, ...)
- `#pragma omp simd reduction(op: var1,...)`

### Problems limiting vector efficiency

#### Hardware under-utilization
- Low usage of zmm registers in AVX512 (Xeon Scalable):  
  - Recompile with `-qopt-zmm-usage=high`  

#### Peel loops significant (i.e. short loops)
Align your data to memory! Choose from (e.g. 32 byte, ymm 256 bit):  
- Compile with `-falalign-loops[=32]`  
- Declare arrays as aligned objects (e.g. static allocations):  
  - C/C++: `__declspec((align(32)) <declaration>;`  
  - Fortran: `cDEC$ ATTRIBUTES ALIGN: 32:: <decl>`  
  - Then ALWAYS tell: `#pragma omp simd aligned(v:32)`

#### Inefficient memory access (strided or irregular)
Run memory access pattern analysis  
Change data layout! 1-D arrays, layout change:  
- Array of Structure → Structure of arrays (AoS→SoA)  
- Use libraries or structures! E.g. Intel® SDLT, or DPC++

* Just use the tool! Why no vectorization? and Recommendations tab  
** C/C++: `#pragma omp simd` → Fortran: `!$omp simd`
Exercise
Optimization of raytrace code

1. Access server, get code, compile it. What does it do?

2. Start with either base or OpenMP version (no MPI).

3. Run one serial/threaded scalar run.

4. Use VTune to thread code, characterize performance, identify main hotspots. Something you can address already? Repeat analysis after each change.

5. Vectorization: use SIMD compilation options. What happens to performance?

6. Use Advisor to identify vectorization issues. What wasn’t vectorized? Why not? Recommendations? What are the main hotspots?

7. How many can you vectorize? Performance?

8. Generate a roofline plot, and analyze it. Repeat analysis after each code change.

9. What happens if you play with tile size? What changes for the threSpeculate and try it out!

TIP #1: You may have to use the CLI version of the tools for the collections (run on compute nodes) and reports (bandwidth limitations).

TIP #2: Raytrace produces a 36 MB image output, slow to interact with from remote. View it by: display image.pnm

- Always check that your modified code gives the right answer (very instructive)!
- Reduce image size when debugging (not recommended for the tool runs)
- or check with: diff old_image.pnm new_image.pnm