Intel Parallel Studio: Vtune

C. Berthelot
Christophe.Berthelot@atos.net

Copyright ©Bull S.A.S. 2016

Laboratório Nacional de Computação Científica
Agenda

- **Introduction**
  Bottleneck
  Gprof

- **Vtune Amplifier XE**
  Introduction
  The Software Optimization Process
  Vtune: GUI
  Sum up
  Labs: Demo
Introduction

Boottleneck

In software engineering, a bottleneck occurs when the capacity of an application or a computer system is severely limited by a single component. The bottleneck has lowest throughput of all parts of the transaction path. As such, system designers will try to avoid bottlenecks and direct effort towards locating and tuning existing bottlenecks. Some examples of possible engineering bottlenecks are: a processor, a communication link, disk IO, etc.

Tracking down bottlenecks (sometimes known as "hot spots" - sections of the code that execute most frequently - i.e. have the highest execution count) is called performance analysis. Reduction is usually achieved with the help of specialized tools, known as performance analyzers or profilers. The objective being to make those particular sections of code perform as fast as possible to improve overall algorithmic efficiency.
Introduction

Gprof

How?

- Compile and link with -p and -g
- You can use GMON_OUT_PREFIX for MPI code
- To summarize the information: gprof -s a.out GMON_OUT_PREFIX.*

Example benchmark bt from NAS (9 tasks)

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>time</th>
<th>seconds</th>
<th>self</th>
<th>calls</th>
<th>s/call</th>
<th>s/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.21</td>
<td>9.10</td>
<td>9.10</td>
<td>67697001</td>
<td>0.00</td>
<td>0.00</td>
<td>binvcrhs_</td>
<td></td>
</tr>
<tr>
<td>17.93</td>
<td>15.32</td>
<td>6.22</td>
<td>603</td>
<td>0.01</td>
<td>0.02</td>
<td>y_solve_cell_</td>
<td></td>
</tr>
<tr>
<td>11.79</td>
<td>19.41</td>
<td>4.09</td>
<td>603</td>
<td>0.01</td>
<td>0.01</td>
<td>z_solve_cell_</td>
<td></td>
</tr>
<tr>
<td>11.60</td>
<td>23.43</td>
<td>4.03</td>
<td>67697001</td>
<td>0.00</td>
<td>0.00</td>
<td>matmul_sub_</td>
<td></td>
</tr>
<tr>
<td>9.86</td>
<td>26.85</td>
<td>3.42</td>
<td>202</td>
<td>0.02</td>
<td>0.02</td>
<td>compute_rhs_</td>
<td></td>
</tr>
<tr>
<td>7.78</td>
<td>29.55</td>
<td>2.70</td>
<td>603</td>
<td>0.00</td>
<td>0.01</td>
<td>x_solve_cell_</td>
<td></td>
</tr>
<tr>
<td>3.20</td>
<td>30.66</td>
<td>1.11</td>
<td>67697001</td>
<td>0.00</td>
<td>0.00</td>
<td>matvec_sub_</td>
<td></td>
</tr>
</tbody>
</table>
Vtune Amplifier XE

- **Introduction**
  - Bottleneck
  - Gprof

- **Vtune Amplifier XE**
  - Introduction
  - The Software Optimization Process
  - Vtune: GUI
  - Sum up
  - Labs: Demo
Vtune Amplifier XE

Advanced level

Find performance bottlenecks with advanced profiling technologies:

▶ Event-Based, System-Wide Sampling with little impact on program execution (typically < 1%).
▶ Call Graph Profiling offers a pictorial view of program flow to help you quickly identify critical functions.
Identify Hotspots

- Identify the Hotspots
- Determine Efficiency
- Identify Architectural Reason for Inefficiency
Vtune Amplifier XE

Optimize issue

Three questions

▶ Why? Why you should be concerned about this potential problem
▶ How? Which profile and metric to use inside Vtune.
▶ What now? Try to give suggestions to try some optimizations
Vtune Amplifier XE

Identify the Hotspots

What?
Hotspots are where your application spends the most time ;-)

Why?
You have to look where you lost a lot of your time

How?
The good event is

\[ \text{CPU_CLK_UNHALTED.THREAD} \]  \tag{1} \]
This counter measures unhalted clockticks on per thread basis. If you use Hyperthreading this event will count 2 ticks for each tick of the CPU's clock.

Vtune

amplxe-cl -collect general-exploration – ./a.out
Vtune Amplifier XE

Determine Efficency of the hotspot

Three ways

- % Pipeline Slots Retired/Cycle
- Changes in CPI (Cycles per Instruction)
- Code examination
Why
This help you to understand how efficiently your application is using the processors

How?

\[
\frac{UOPS\_RETIRED\_RETIRE\_SLOTS}{CPU\_CLK\_UNHALTED\_THREAD}
\]

What Now, for a given hotspot?

- If > 90% retiring (0.9 or higher) is good. Go to efficiency method 3 (code examination)
- Between 50 and 90% for client apps investigating stall reduction
- Less than 60% for server apps consider stall reduction
Vtune Amplifier XE

Efficiency: Changes in Cycles per Instruction: CPI

Why?
A measure of efficiency that can be used to compare two runs.

How?
General exploration profile (snb-general-exploration)

\[
\frac{CPU_{-}CLK_{-}UNHALTED._THREAD}{INST_{-}RETIRE_{-}ANY}
\]

What now?
- CPI is a ratio, if the code size changes for a binary, CPI will change. In general, if CPI reduces as a result of optimizations, that is good, and if it increases, that is bad.
- Optimized code may actually lower the CPI, and increase stall % but it will increase the performance. CPI is just a general efficiency metric the real measure of efficiency is work taking less time.
Vtune Amplifier XE

Efficiency Method 3: Code Examination

Why?
The two first methods look how long it takes instruction to execute. The other type of inefficiency is to execute too many instructions.

How?
With VTune capability to mixt source and disassembly with viewer.

What now?
This method involves looking at the disassembly to make sure the most efficient instruction streams are generated. This can be complex and can require an expert knowledge of the Intel instruction set and compiler technology.
Vtune Amplifier XE

First step

Load Env

- source . /opt/intel/parallel_studio_xe_YYYY.XX.YY/psxevars.sh
- Run amplxe-gui
Vtune Amplifier XE

New project
Vtune Amplifier XE

Select Target and options

Choose Target and Analysis Type

Launch Application
Specify and configure your analysis target: an application or a script to execute. Press F1 for more details.

No application executable (target) file specified.

Application:
Application parameters:
Use application directory as working directory

Working directory:

User-defined environment variables:

Managed code profiling mode: Auto

Automatically resume collection after (sec):

Automatically stop collection after (sec):

Advanced
Vtune Amplifier XE

Select Target and options (advanced)
Vtune Amplifier XE

Select a new analysis
Vtune Amplifier XE

Command line to use inside batch

```
/opt/intel/vtune_amplifier_xe_2016.3.0.463186/bin64/amplxe-cl -collect advanced-hotspots -app-working-dir /home_nfs/berthelc/TPFORMATION/NPB3.2.1/NPB3.2-SER/bin -- /home_nfs/berthelc/TPFORMATION/NPB3.2.1/NPB3.2-SER/bin/cg.B
```

- Use -collect-with action
- Hide knobs with default values
First windows after the run

Elapsed Time: 100.319s

CPU Time:
- Instructions Retired: 126,570,660,000
- CPI Rate: 1.967

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

- CPU Frequency Ratio: 1.104
- Total Thread Count: 2
- Paused Time: 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>coni Grad</td>
<td>cg8</td>
<td>94.686s</td>
</tr>
<tr>
<td>MAIN</td>
<td>cg8</td>
<td>4.481s</td>
</tr>
<tr>
<td>[Outside any known module]</td>
<td>[Unknown]</td>
<td>0.306s</td>
</tr>
<tr>
<td>randic</td>
<td>cgB</td>
<td>0.025s</td>
</tr>
<tr>
<td>_errno_location</td>
<td>libpthread-2.12.so</td>
<td>0.001s</td>
</tr>
<tr>
<td>[Others]</td>
<td>N/A*</td>
<td>0.003s</td>
</tr>
</tbody>
</table>

CPU Usage Histogram

The histogram displays a percentage of the wall time the specific number of CPU's were running circuitously. Spin and Overhead time adds to the idle CPU.
Vtune Amplifier XE

Hotspots
## Vtune Amplifier XE

### Low level: ASM view

![Vtune Amplifier XE interface](image)

<table>
<thead>
<tr>
<th>Address</th>
<th>Sou., Line</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x405be2</td>
<td>mulpdx 0x8(%rsi,%rcx,8), %xmm10</td>
<td></td>
</tr>
<tr>
<td>0x405be9</td>
<td>mulpdx 0x8(%rsi,%rcx,8), %xmm11</td>
<td></td>
</tr>
<tr>
<td>0x405bf0</td>
<td>mulpdx 0x18(%rsi,%rcx,8), %xmm12</td>
<td></td>
</tr>
<tr>
<td>0x405bf7</td>
<td>addpd %xmm10, %xmm8</td>
<td></td>
</tr>
<tr>
<td>0x405bfc</td>
<td>addpd %xmm11, %xmm7</td>
<td></td>
</tr>
<tr>
<td>0x405c01</td>
<td>addpd %xmm12, %xmm6</td>
<td></td>
</tr>
<tr>
<td>0x405c06</td>
<td>movhpdq -0x8(%r10,%rbp,8), %xmm13</td>
<td></td>
</tr>
<tr>
<td>0x405c0d</td>
<td>mulpdx 0x28(%rsi,%rcx,8), %xmm13</td>
<td></td>
</tr>
<tr>
<td>0x405c14</td>
<td>add %edx, %rcx</td>
<td></td>
</tr>
<tr>
<td>0x405c18</td>
<td>addpd %xmm13, %xmm5</td>
<td></td>
</tr>
<tr>
<td>0x405c1d</td>
<td>cmp %r13, %rcx</td>
<td></td>
</tr>
<tr>
<td>0x405c20</td>
<td>jb 0x405b8a &lt;Block 43&gt;</td>
<td></td>
</tr>
<tr>
<td>0x405c26</td>
<td>Block 44;</td>
<td></td>
</tr>
<tr>
<td>0x405c26</td>
<td>addpd %xmm7, %xmm8</td>
<td></td>
</tr>
<tr>
<td>0x405c2b</td>
<td>addpd %xmm5, %xmm6</td>
<td></td>
</tr>
<tr>
<td>0x405c2f</td>
<td>addpd %xmm6, %xmm5</td>
<td></td>
</tr>
<tr>
<td>0x405c34</td>
<td>movaps %xmm8, %xmm5</td>
<td></td>
</tr>
</tbody>
</table>

**CPU Time**

<table>
<thead>
<tr>
<th>Effective Time by Utilization</th>
<th>Spin Time</th>
<th>Ov. Time</th>
<th>Ins... Re...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>6.450s</td>
<td>0s</td>
<td>7.1..</td>
</tr>
<tr>
<td>Poor</td>
<td>4.441s</td>
<td>0s</td>
<td>3.0..</td>
</tr>
<tr>
<td>Ok</td>
<td>4.793s</td>
<td>0s</td>
<td>3.4..</td>
</tr>
<tr>
<td>Ideal</td>
<td>6.149s</td>
<td>0s</td>
<td>4.6..</td>
</tr>
<tr>
<td>Over</td>
<td>0.184s</td>
<td>0s</td>
<td>90..</td>
</tr>
<tr>
<td></td>
<td>0.999s</td>
<td>0s</td>
<td>860..</td>
</tr>
<tr>
<td></td>
<td>1.811s</td>
<td>0s</td>
<td>1.6..</td>
</tr>
<tr>
<td></td>
<td>3.928s</td>
<td>0s</td>
<td>5.1..</td>
</tr>
<tr>
<td></td>
<td>7.316s</td>
<td>0s</td>
<td>6.3..</td>
</tr>
</tbody>
</table>
Introduction
You can use vtune with Intel MPI. It does not work with all MPI. For other application see paper Analyzing MPI programs with Intel VTune Amplifier XE and Intel Inspector XE tools.

How

```bash
mpirun -n <N> -gtools "<abbr>-cl -r my_result -collect <analysis type>:MIRANK" my_app [my_app_options]
```

The list of analysis types available can be viewed using amplxe-cl -help collect. The most simple to start with vtune is to use hotspot as analysis.
**Vtune Amplifier XE**

**Positives points**
- Vtune: easy to use (first level)
- First level of profiling you don’t have to know information about processor, you have to believe the tool
- Vtune works with MPI and with slurm

**Difficulties points**
- To extract all informations have to understand μ-arch
- To extract all informations have to known some information about ratio or build you own.
Use module to set your env

- load parallel studio XE : source /opt/intel/parallel_studio_xe_YYY.XX.ZZZ/psxevars.sh
- Set export VISUAL=gedit
- Extract
  /opt/intel/parallel_studio_xe_YYYY.XX.ZZZ/vtune_amplifier_xe_YYYY/samples/en/C++/tachyon_vtune_amp_xe.tgz
- Compile : make
Vtune Amplifier XE

Labs: Hotspots

First Run

- Run amplxe-gui
- New Project
- New Analysis (Hotspot)
- Fin hotspot
  - Create New Project
  - load binary :tachyon_find_hotspots
  - parameter data/balls.dat
  - select hotspot
  - Run application
Vtune Amplifier XE

Labs: Hotspots

Code modification

- Edit file
- Modification of memory access
- Compile (make)
- New Analysis (Hotspot)
- Run application
Vtune Amplifier XE

Labs: Hotspots

Compare

- Load 2 files
- Compare results
Vtune Amplifier XE

Vtune and MPI

▶ Go on TP_HPCToolkit/NPB3.2.1/NPB3.2-MPI
▶ Compile  make CG CLASS=B NPROCS=16
▶ Run code with vtune on rank 0:
  mpirun -gtool "amplxe-cl -collect hpc-performance -r result:0" -n 16 ./cg.B.16
▶ Load result inside Vtune GUI
Vtune Amplifier XE

Vtune and OpenMP

- Go on TP_HPCToolkit/NPB3.2.1/NPB3.2-OMP
- Compile make CG CLASS=B
- Run code with numact all on node 0, and CPU on node 0 / meme en node 1
- Load result inside GUI
COPYRIGHT NOTICE

- ©Bull. All rights reserved
  - Users Restricted Rights - Use, duplication or disclosure restricted.
  - Any copy of these documents should keep all copyright, logos and other proprietary notices contained herein.
  - This publication may include technical inaccuracies or typographical errors.
  - This publication is provided "AS IS" without any warranty either expressed or implied including but not limited to the implied warranties of merchantabilities or fitness of the described product.
  - Course Material Licensing Terms : No sublicensing rights.
  - For other licensing needs, please contact Bull