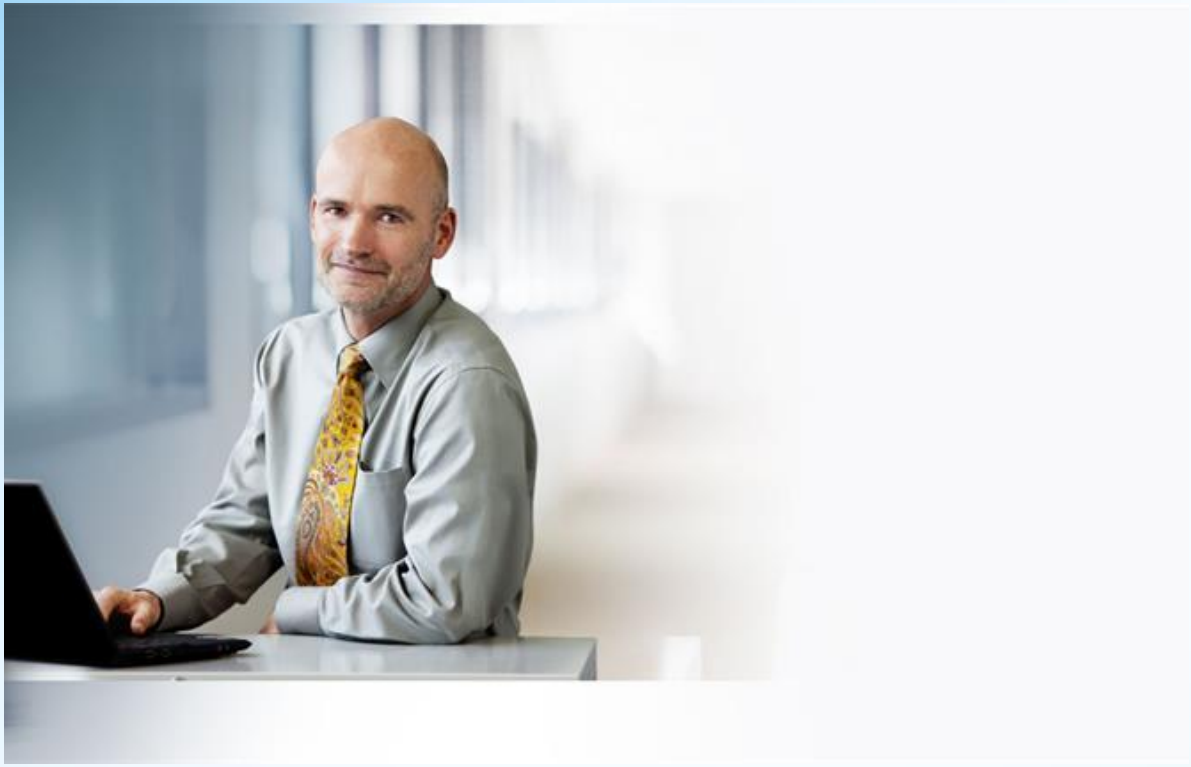


# A Performance Tuning Methodology: From the System Down to the Hardware - Introduction

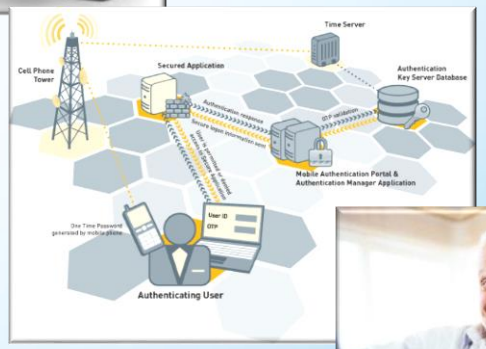
Jackson Marusarz
Intel Corporation
ATPESC 2014

# Why performance profiling?

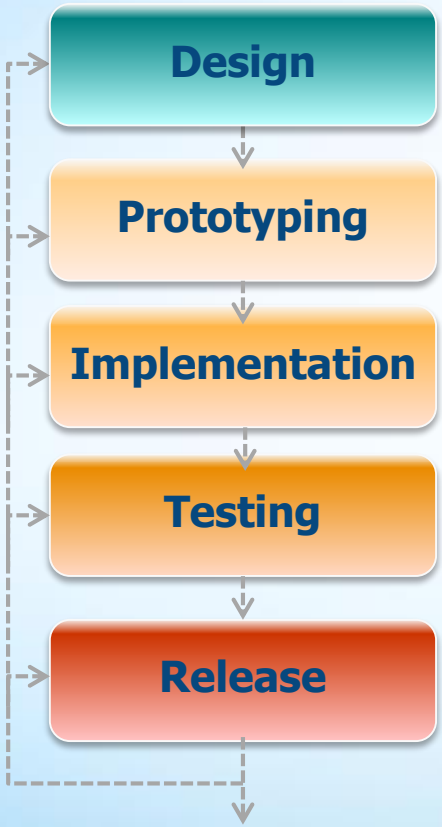
**Project performance tuning for:**

- Reducing direct compute time costs
- Decreasing indirect costs
- Better user/customer experience

If you are not in that business, don't bother




# Project development cycle and performance analysis




Think performance wise (app/sys level)

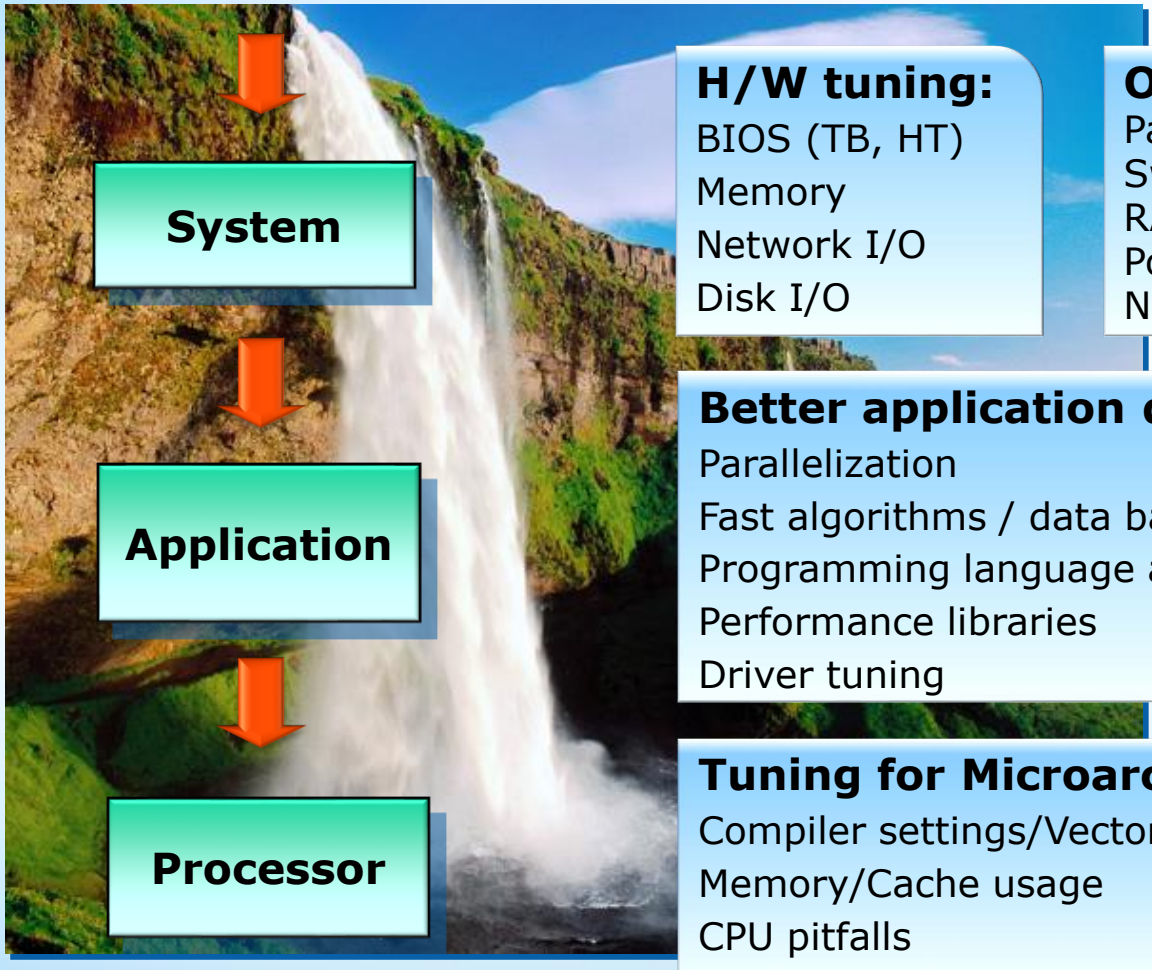
Choose perf. effective solutions

Apply perf. optimization and check results

Add perf. regression phase to test stage

Collect and analyze perf. related complaints from users of your product

# Optimization: A Top-down Approach




**H/W tuning:**
BIOS (TB, HT)
Memory
Network I/O
Disk I/O

**OS tuning:**
Page size
Swap file
RAM Disk
Power settings
Network protocols

**Better application design:**
Parallelization
Fast algorithms / data bases
Programming language and RT libs
Performance libraries
Driver tuning

**Tuning for Microarchitecture:**
Compiler settings/Vectorization
Memory/Cache usage
CPU pitfalls

OS, System **Expertise** SW/uArch

https://software.intel.com/en-us/articles/de-mystifying-software-performance-optimization

# Performance profiling tools
## Level wise selection

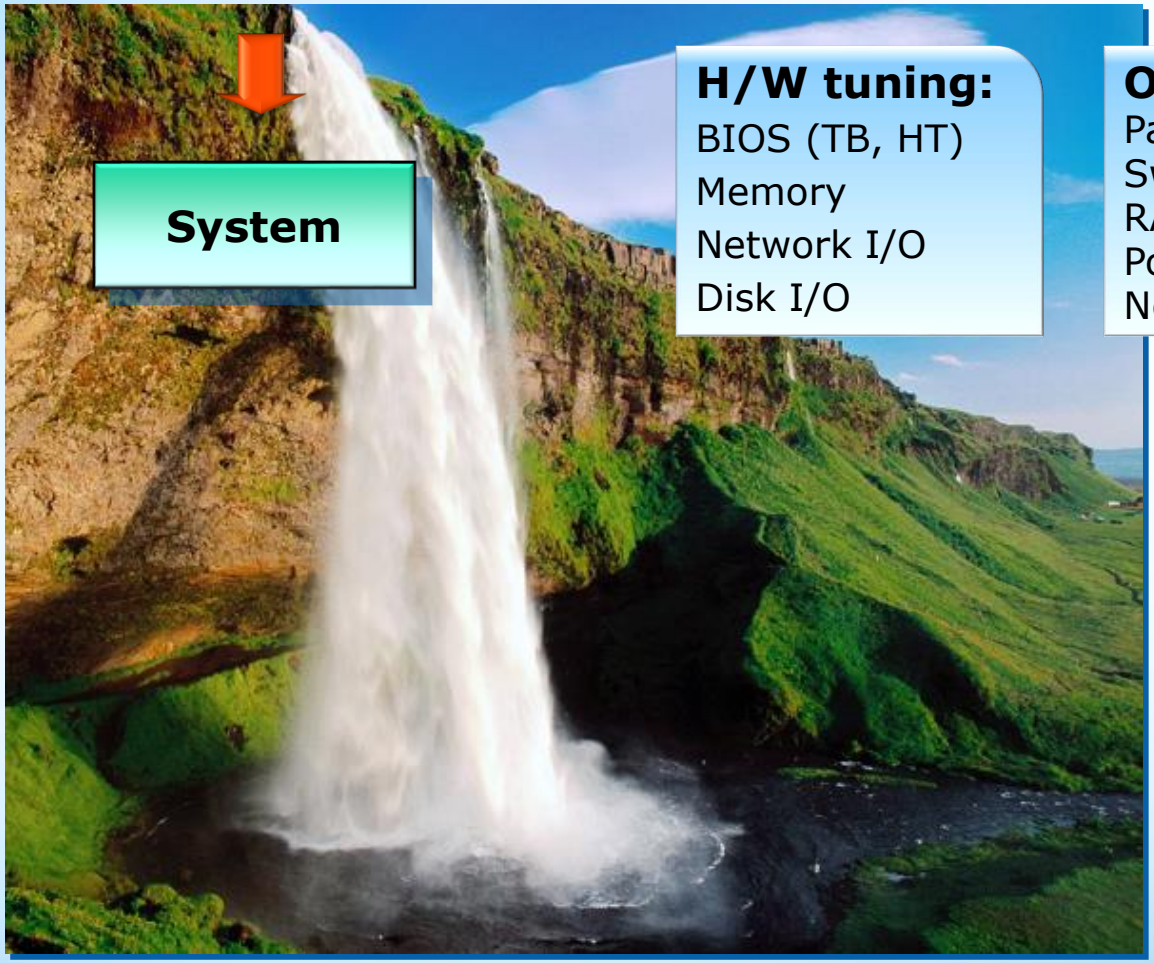
| | | |
|---|---|---|
| **System** | System profiler | OS embedded |
| | Universal (for OS, HW) | Windows: Perf mon, Proc mon |
| | Proprietary (OS+HW) | Linux: top, vmstat, OProfile |
| **Application** | Supported languages | Windows: WPT, Xperf, VTune |
| IDE based | .Net/C#, Java | Managed: .Net, Java tools, VTune |
| | Python, Java Script, HTML | Linux: gprof, Valgrind, Google perftools, Crxprof, VTune |
| Command Line | C, C++, Fortran | |
| **Microarchitcture** | Provided by CPU/Platform manufacturer | |

# Performance profiling tools
## Level wise selection

| | | |
|---|---|---|
| **System** | System profiler | OS embedded |
| | Universal (for OS, HW) | Windows: Perf mon, Proc mon |
| | Proprietary (OS+HW) | Linux: top, vmstat, OProfile |
| **Application** | Supported languages | Windows: WPT, Xperf, VTune |
| IDE based | .Net/C#, Java | Managed: .Net, Java tools, VTune |
| | Python, Java Script, HTML | Linux: gprof, Valgrind, Google perftools, Crxprof, VTune |
| Command Line | C, C++, Fortran | |
| **Microarchitcture** | Provided by CPU/Platform manufacturer | |

Tools are essential for efficient performance analysis.

# Optimization: A Top-down Approach




**H/W tuning:**
BIOS (TB, HT)
Memory
Network I/O
Disk I/O

**OS tuning:**
Page size
Swap file
RAM Disk
Power settings
Network protocols

OS, System

# System Tuning

**Who: System Administrators, Performance Engineers, Machine Owners, etc…**

**How:**

- Benchmarks
  - Stream: www.cs.virginia.edu/stream/
  - Numerous FLOPS benchmarks
  - Network/MPI Benchmarks: www.intel.com/go/imb
  - <insert your favorite here>
- Tools
  - vmstat, top, sysprof, iostat, sar, Task Manager, etc…
  - Many vendor/platform specific tools

- Fixes
  - Upgrade Hardware - **$$$**
  - Check BIOS and OS configurations
    - Prefetchers, NUMA, Memory Configuration, Power Management, SMT

# System Tuning

**Who: System Administrators, Performance Engineers, Machine Owners, etc…**

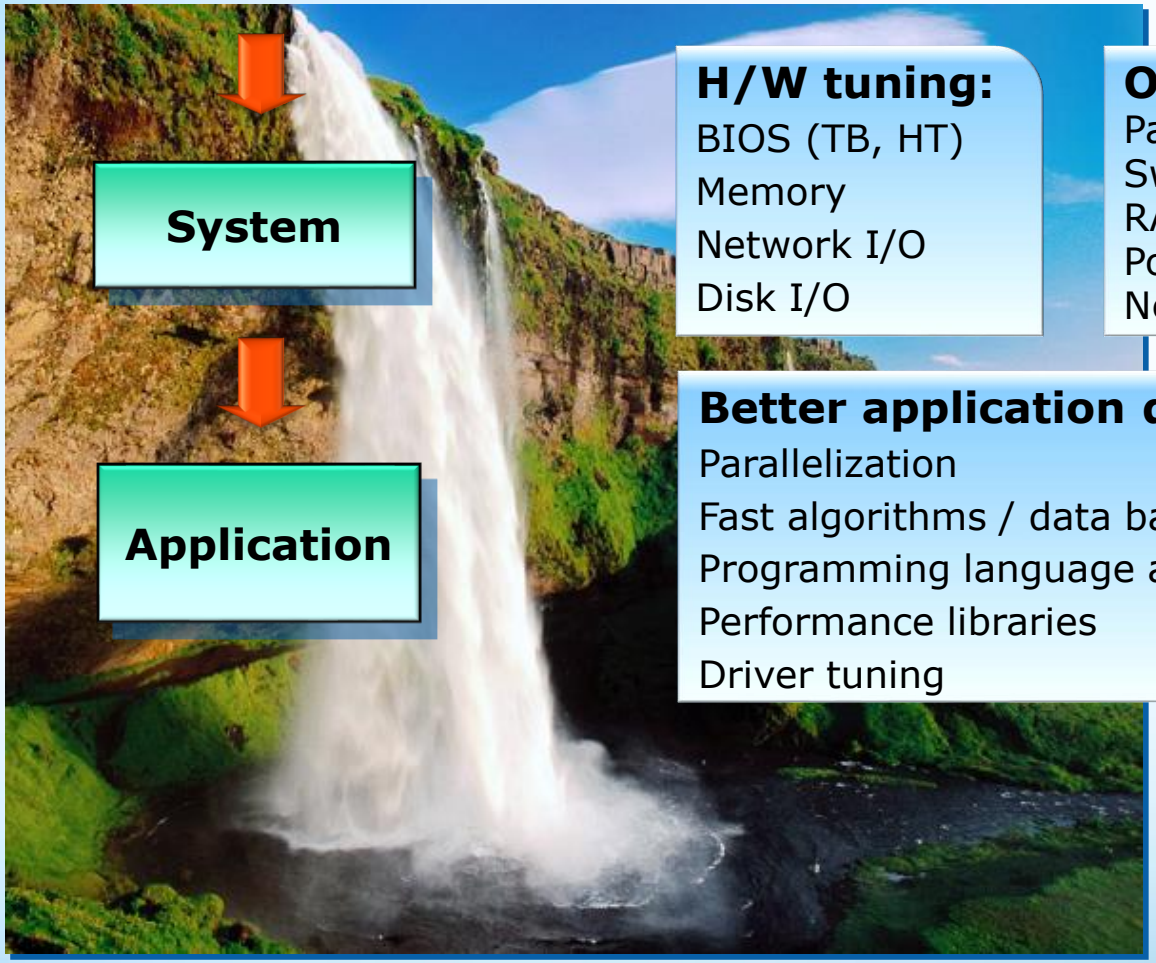
**How:**

- Benchmarks
  - Stream: www.cs.virginia.edu/stream/
  - Numerous FLOPS benchmarks
  - Network/MPI Benchmarks: www.intel.com/go/imb
  - <insert your favorite here>
- Tools
  - vmstat, top, sysprof, iostat, sar, Task Manager, etc…
  - Many vendor/platform specific tools

- Fixes
  - Upgrade Hardware - **$$$**
  - Check BIOS and OS configurations
    - Prefetchers, NUMA, Memory Configuration, Power Management, SMT

This is often outside the capabilities of most users

# Optimization: A Top-down Approach




**H/W tuning:**
BIOS (TB, HT)
Memory
Network I/O
Disk I/O

**OS tuning:**
Page size
Swap file
RAM Disk
Power settings
Network protocols

**Better application design:**
Parallelization
Fast algorithms / data bases
Programming language and RT libs
Performance libraries
Driver tuning

OS, System **Expertise**

# Application Tuning

## Who: Software Developers, Performance Engineers, Domain Experts

**How:**

- Workload selection
  - Repeatable results
  - Steady state
- Define Metrics and Collect Baseline
  - Wall-clock time, FLOPS, FPS
  - <insert your metric here>
- Identify Hotspots
  - Focus effort where it counts
  - Use Tools

- Determine inefficiencies
  - Is there parallelism?
  - Are you memory bound?
  - Will better algorithms or programming languages help?

This step often requires some knowledge of the application and its algorithms

# Application Tuning
## Find Hotspots

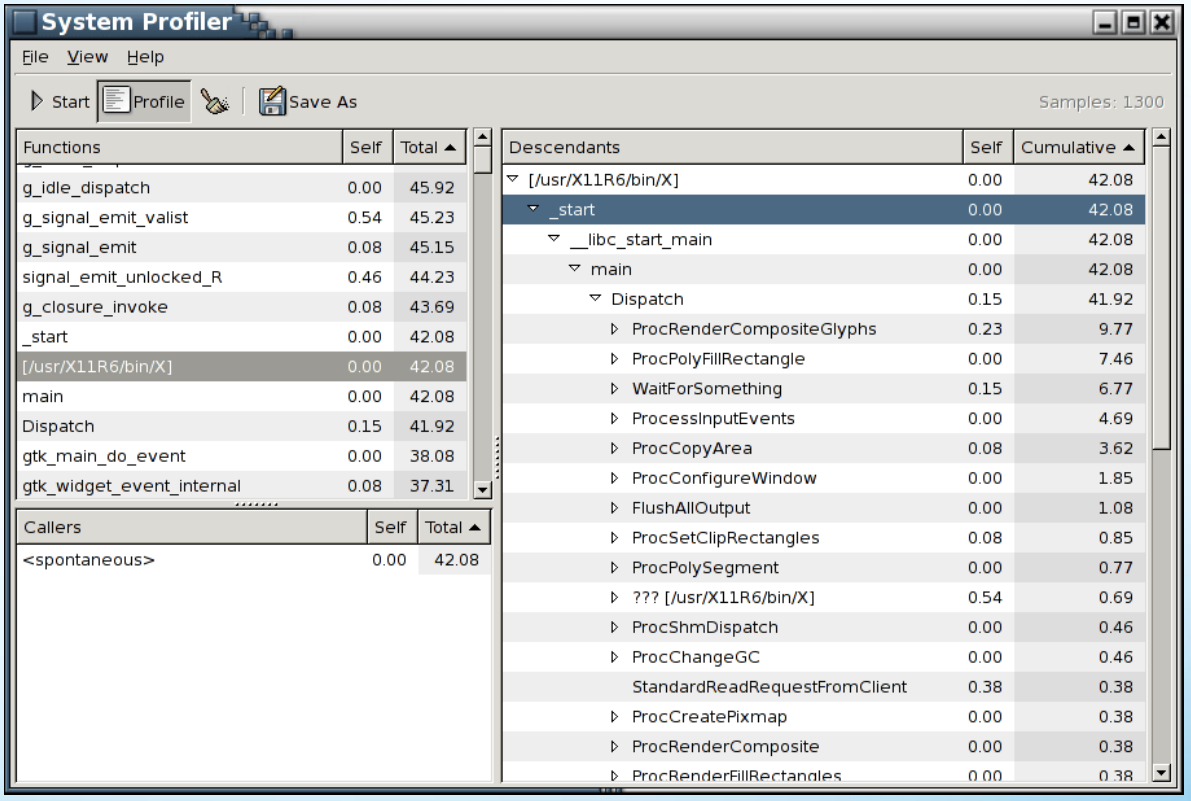
- This could be at the module, function, or source code level
- Determine your own granularity

```
$ opreport --exclude-dependent --demangle=smart --symbols `which lyx`
CPU: PIII, speed 863.195 MHz (estimated)
Counted CPU_CLK_UNHALTED events (clocks processor is not halted) with a unit mask of 0x00 (No unit mask)
vma      samples  %           symbol name
081ec974 5016      8.5096     _Rb_tree<unsigned short, pair<unsigned short const, int>,  unsigned short
0810c4ec 3323      5.6375     Paragraph::getFontSettings(BufferParams const&, int) const
081319d8 3220      5.4627     LyXText::getFont(Buffer const*, Paragraph*, int) const
080e45d8 3011      5.1082     LyXFont::realize(LyXFont const&)
080e3d78 2623      4.4499     LyXFont::LyXFont()
081255a4 1823      3.0927     LyXText::singleWidth(BufferView*, Paragraph*, int, char) const
080e3cf0 1804      3.0605     operator==(LyXFont::FontBits const&, LyXFont::FontBits const&)
081128e0 1729      2.9332     Paragraph::Pimpl::getChar(int) const
081ed020 1380      2.3412     font_metrics::width(char const*, unsigned, LyXFont const&)
08110d60 1310      2.2224     Paragraph::getChar(int) const
081ebc94 1227      2.0816     qfont_loader::getfontinfo(LyXFont const&)
...
```

oprofile: http://oprofile.sourceforge.net/

# Application Tuning
## Find Hotspots

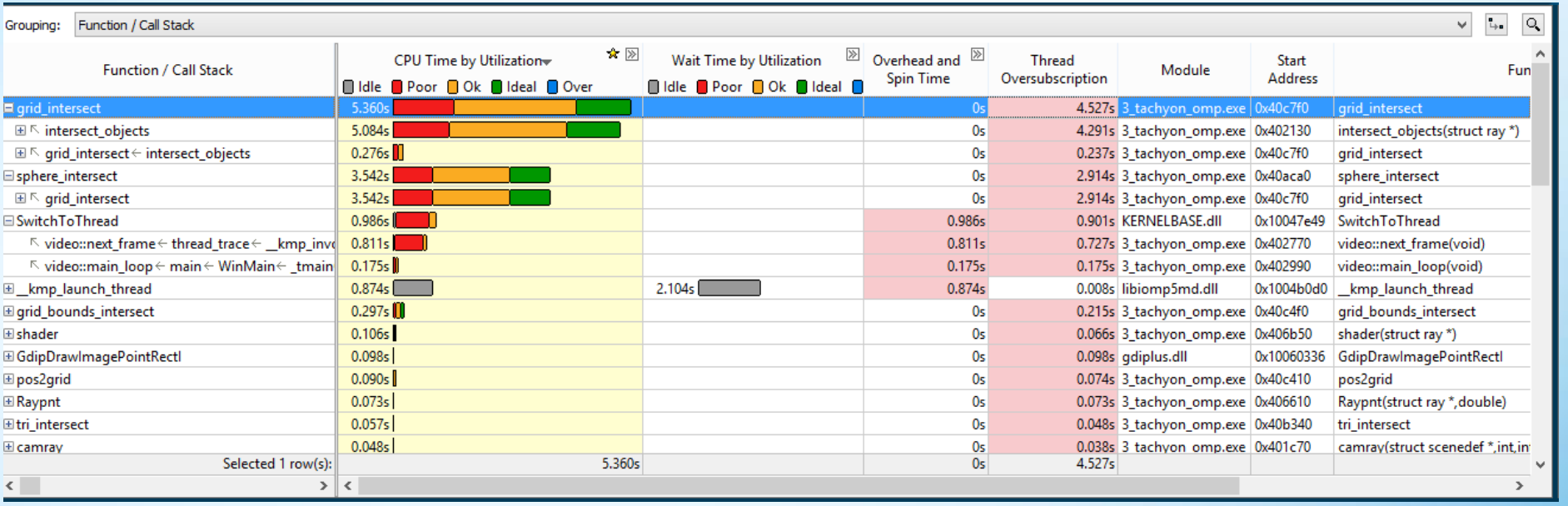
- This could be at the module, function, or source code level
- Determine your own granularity




sysprof: http://sysprof.com

# Application Tuning
## Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity

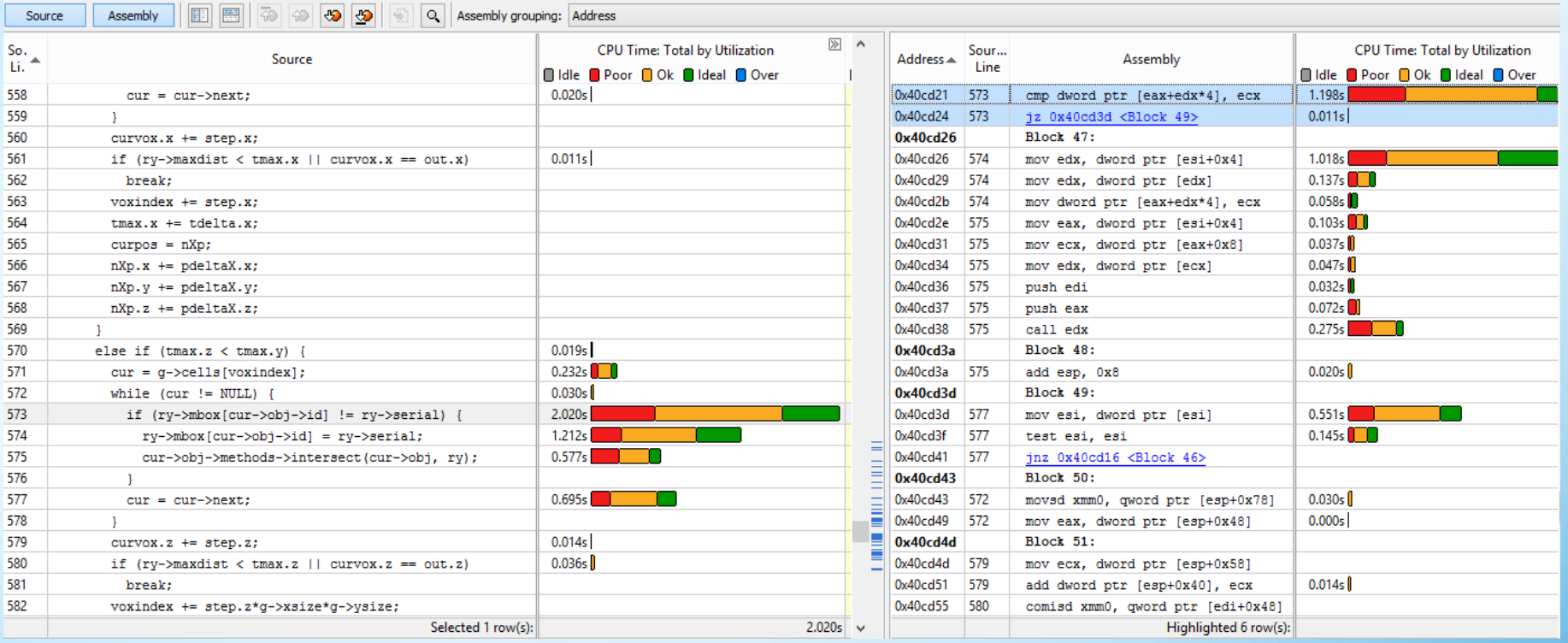



| Function / Call Stack | CPU Time by Utilization (Idle Poor Ok Ideal Over) | Wait Time by Utilization (Idle Poor Ok Ideal) | Overhead and Spin Time | Thread Oversubscription | Module | Start Address | Fun |
|---|---|---|---|---|---|---|---|
| grid_intersect | 5.360s | | 0s | 4.527s | 3_tachyon_omp.exe | 0x40c7f0 | grid_intersect |
| intersect_objects | 5.084s | | 0s | 4.291s | 3_tachyon_omp.exe | 0x402130 | intersect_objects(struct ray *) |
| grid_intersect ← intersect_objects | 0.276s | | 0s | 0.237s | 3_tachyon_omp.exe | 0x40c7f0 | grid_intersect |
| sphere_intersect | 3.542s | | 0s | 2.914s | 3_tachyon_omp.exe | 0x40aca0 | sphere_intersect |
| grid_intersect | 3.542s | | 0s | 2.914s | 3_tachyon_omp.exe | 0x40c7f0 | grid_intersect |
| SwitchToThread | 0.986s | | 0.986s | 0.901s | KERNELBASE.dll | 0x10047e49 | SwitchToThread |
| video::next_frame ← thread_trace ← __kmp_invc | 0.811s | | 0.811s | 0.727s | 3_tachyon_omp.exe | 0x402770 | video::next_frame(void) |
| video::main_loop ← main ← WinMain ← _tmain | 0.175s | | 0.175s | 0.175s | 3_tachyon_omp.exe | 0x402990 | video::main_loop(void) |
| __kmp_launch_thread | 0.874s | 2.104s | 0.874s | 0.008s | libiomp5md.dll | 0x1004b0d0 | __kmp_launch_thread |
| grid_bounds_intersect | 0.297s | | 0s | 0.215s | 3_tachyon_omp.exe | 0x40c4f0 | grid_bounds_intersect |
| shader | 0.106s | | 0s | 0.066s | 3_tachyon_omp.exe | 0x406b50 | shader(struct ray *) |
| GdipDrawImagePointRectI | 0.098s | | 0s | 0.098s | gdiplus.dll | 0x10060336 | GdipDrawImagePointRectI |
| pos2grid | 0.090s | | 0s | 0.074s | 3_tachyon_omp.exe | 0x40c410 | pos2grid |
| Raypnt | 0.073s | | 0s | 0.073s | 3_tachyon_omp.exe | 0x406610 | Raypnt(struct ray *,double) |
| tri_intersect | 0.057s | | 0s | 0.048s | 3_tachyon_omp.exe | 0x40b340 | tri_intersect |
| camray | 0.048s | | 0s | 0.038s | 3_tachyon_omp.exe | 0x401c70 | camray(struct scenedef *,int,in |
| Selected 1 row(s): | 5.360s | | 0s | 4.527s | | | |

Intel® VTune™ Amplifier XE: http://intel.ly/vtune-amplifier-xe

# Application Tuning
## Find Hotspots

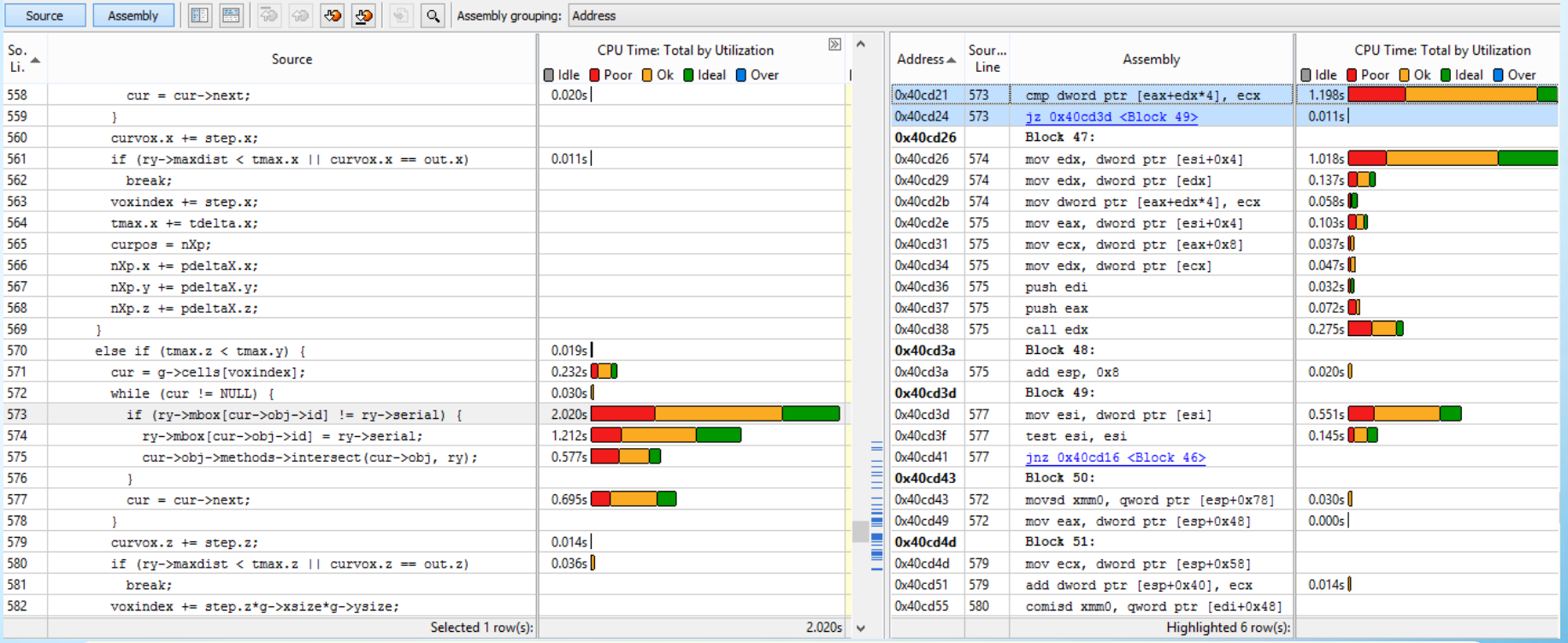
- This could be at the module, function, or source code level
- Determine your own granularity




Intel® VTune™ Amplifier XE: http://intel.ly/vtune-amplifier-xe

# Application Tuning
## Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity




This may reinforce your understanding of the application but often reveals surprises

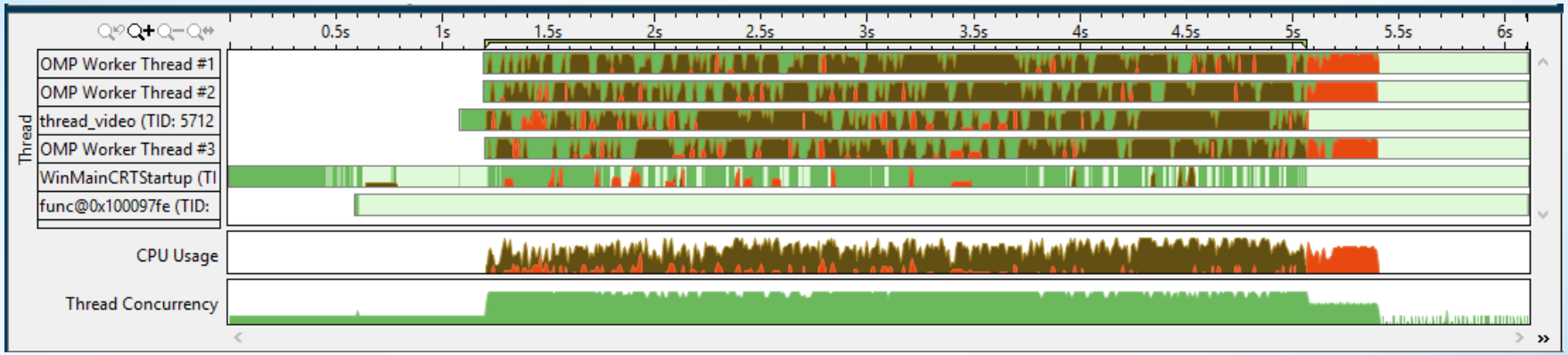
# Application Tuning
## Resource Utilization

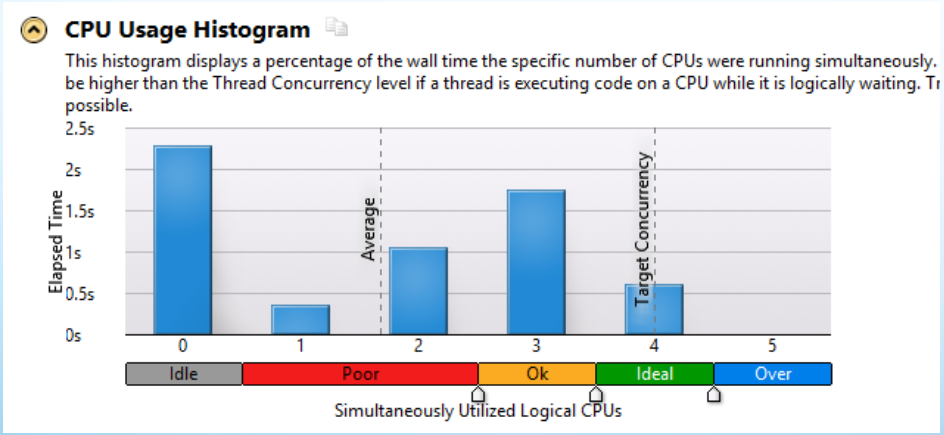
- Is the application parallel?
- Multi-thread vs. Multi-process
- Memory Bound?

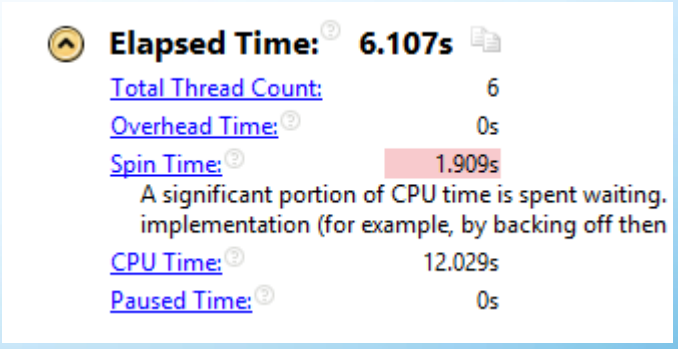
```
last pid: 86494;  load averages:  0.83,  0.65,  0.69   up 67+22:48:43  14:44:15
227 processes: 1 running, 224 sleeping, 2 zombie
CPU: 20.2% user,  0.0% nice,  6.5% system,  0.2% interrupt, 73.1% idle
Mem: 1657M Active, 1868M Inact, 273M Wired, 190M Cache, 112M Buf, 11M Free
Swap: 4500M Total, 249M Used, 4251M Free, 5% Inuse

  PID USERNAME  THR PRI NICE   SIZE    RES STATE  C   TIME   WCPU COMMAND
86460 www         1   4    0   150M 30204K accept 1   0:02 11.18% php-cgi
86458 www         1   4    0   150M 29912K accept 0   0:02  8.98% php-cgi
86463 pgsql       1   4    0   949M    99M sbwait 1   0:01  7.96% postgres
85885 www         1   4    0   150M 35204K accept 2   0:07  7.57% php-cgi
85274 www         1   4    0   149M 40868K sbwait 3   0:27  5.18% php-cgi
85267 www         1   4    0   151M 40044K sbwait 2   0:33  4.59% php-cgi
85884 www         1   4    0   150M 41584K accept 2   0:14  4.59% php-cgi
85887 pgsql       1   4    0   951M   128M sbwait 1   0:04  4.20% postgres
85886 pgsql       1   4    0   949M   161M sbwait 0   0:08  3.37% postgres
86459 pgsql       1   4    0   949M 75960K sbwait 2   0:01  3.37% postgres
85279 pgsql       1   4    0   950M   192M sbwait 2   0:14  2.39% postgres
85269 pgsql       1   4    0   950M   199M sbwait 1   0:19  2.20% postgres
85268 www         1   4    0   152M 44356K sbwait 2   0:32  1.17% php-cgi
85273 pgsql       1   4    0   950M   215M sbwait 0   0:19  1.17% postgres
97082 pgsql       1  44    0 26020K  6832K select 0  46:55  0.00% postgres
  892 root        1   4    0  3160K     8K -      2  13:33  0.00% nfsd
 1796 root        1  44    0 19780K 13660K select 3  12:43  0.00% Xvfb
```

# Application Tuning
## Resource Utilization

- Is the application parallel?

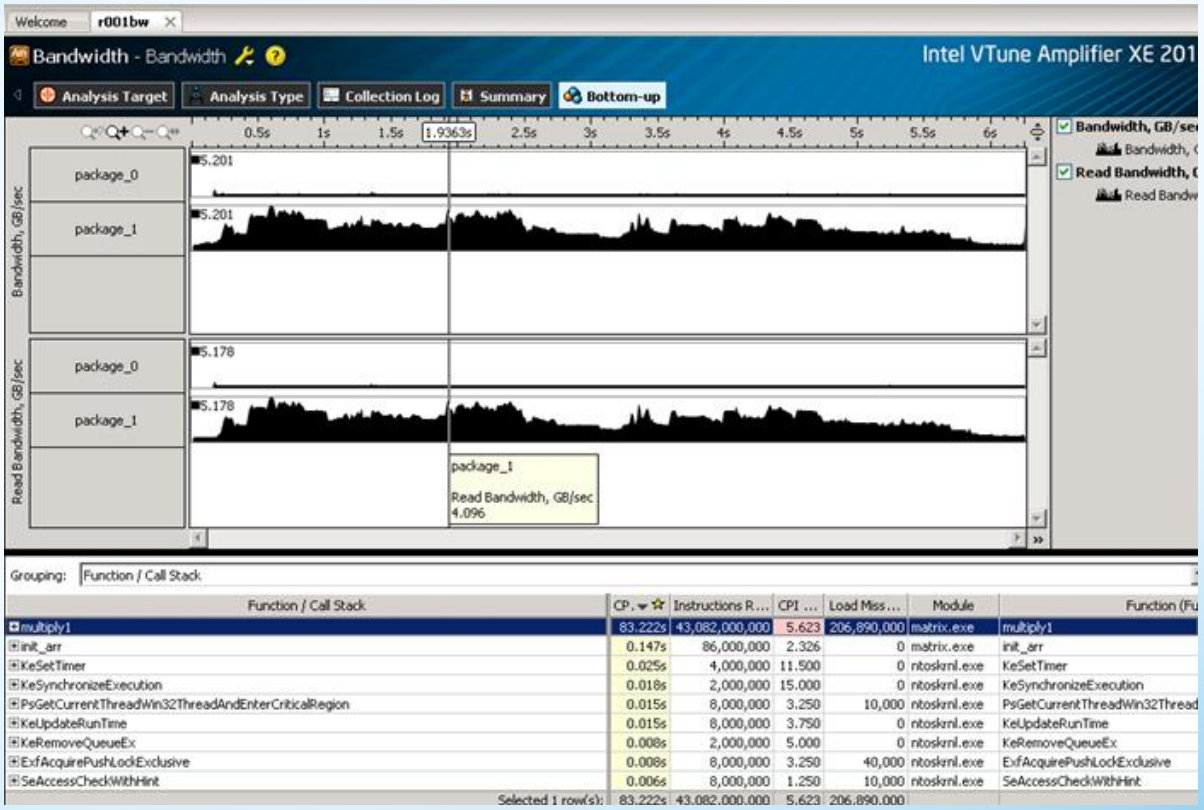


# Application Tuning
## Resource Utilization

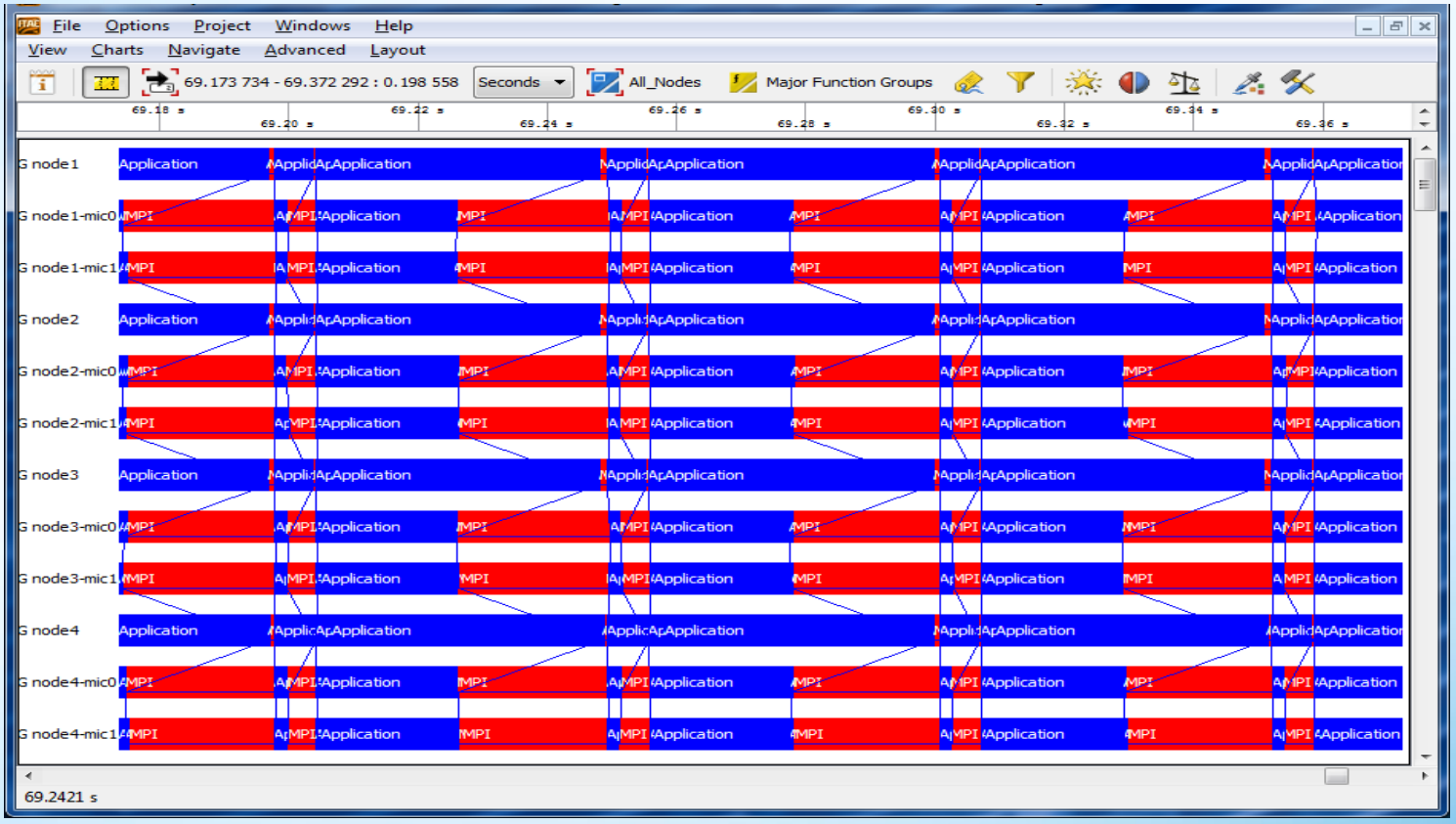
- Memory Bound?




- Know your max theoretical memory bandwidth

# Application Tuning
## Resource Utilization

MPI applications have added communication complexity




Intel® Trace Analyzer and Collector: http://intel.ly/traceanalyzer-collector

# Application Tuning
## What's Next?

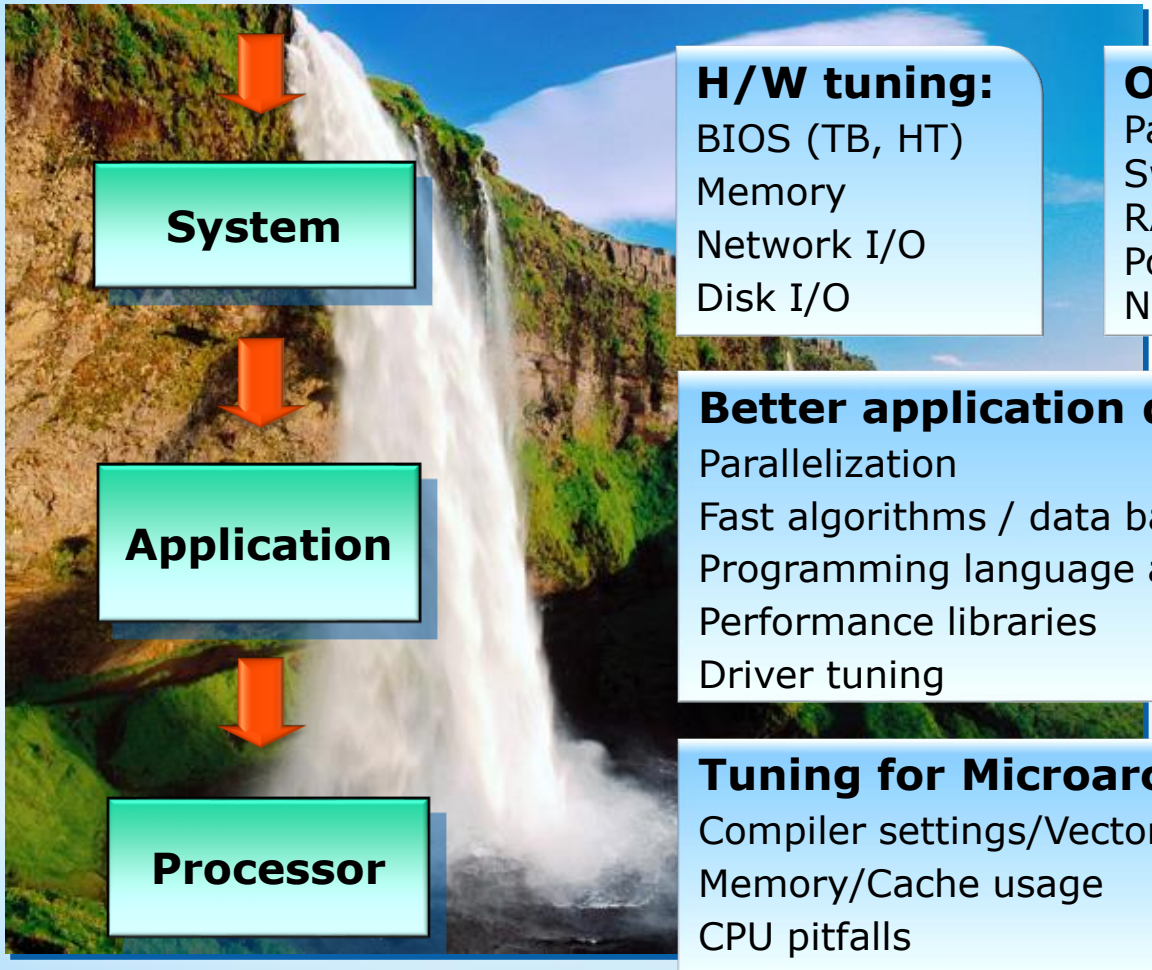
- If your Hotspots are common algorithms:
  - Look for optimized libraries

- If your Hotspots are uncommon:
  - Compiler optimizations
  - Expert analysis and refactoring of an algorithm
    - The opposite of "low-hanging fruit"
  - Deeper analysis of hardware performance
    - More on this later

- If the system is underutilized:
  - Add parallelism - multi-thread or multi-process
    - OpenMP, TBB, Cilk, MPI, etc…

- Tools can help you determine where to look and may identify some issues.
- Some tools may provide suggestions for fixes.
- In the end – the developer and/or expert has to make the changes and decisions – there is no sliver bullet.

# Optimization: A Top-down Approach




**H/W tuning:**
BIOS (TB, HT)
Memory
Network I/O
Disk I/O

**OS tuning:**
Page size
Swap file
RAM Disk
Power settings
Network protocols

**Better application design:**
Parallelization
Fast algorithms / data bases
Programming language and RT libs
Performance libraries
Driver tuning

**Tuning for Microarchitecture:**
Compiler settings/Vectorization
Memory/Cache usage
CPU pitfalls

OS, System **Expertise** SW/uArch

# Microarchitecture Tuning

**Who: ~~Architecture Experts~~**
**Software Developers, Performance Engineers, Domain Experts**

**How:**

- Use architecture specific hardware events
- Use predefined metrics and best known methods
  - Often hardware specific
  - (Hopefully) provided by the vendor

- Tools make this possible for the non-expert
  - Linux perf
  - Intel® VTune™ Amplifier XE

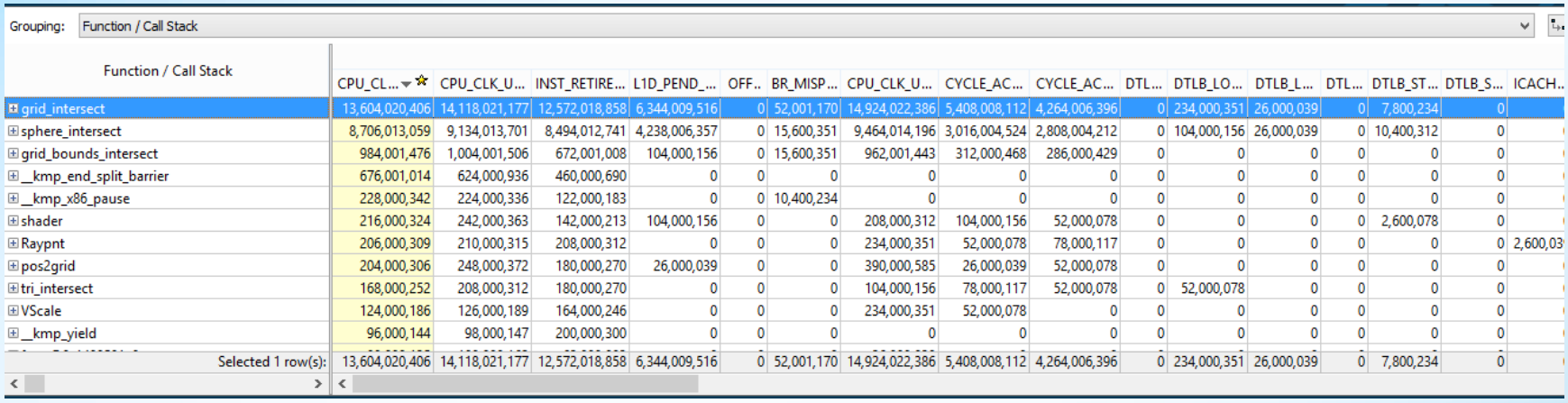
- Follow the Top-Down Characterization
  - Locate the hardware bottlenecks
  - Whitepaper here: https://software.intel.com/en-us/articles/how-to-tune-applications-using-a-top-down-characterization-of-microarchitectural-issues

Now we're getting into Intel specific tuning

# Introduction to Performance Monitoring Unit (PMU)

- Registers on Intel CPUs to count architectural events
  - E.g. Instructions, Cache Misses, Branch Mispredict
- Events can be counted or sampled
  - Sampled events include Instruction Pointer
- Raw event counts are difficult to interpret
  - Use a tool like VTune or Perf with predefined metrics

# Raw PMU Event Counts vs Metrics



Grouping: Function / Call Stack

| Function / Call Stack | CPU_CL... | CPU_CLK_U... | INST_RETIRE... | L1D_PEND_... | OFF.. | BR_MISP... | CPU_CLK_U... | CYCLE_AC... | CYCLE_AC... | DTL... | DTLB_LO... | DTLB_L... | DTL... | DTLB_ST... | DTLB_S... | ICACH.. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| grid_intersect | 13,604,020,406 | 14,118,021,177 | 12,572,018,858 | 6,344,009,516 | 0 | 52,001,170 | 14,924,022,386 | 5,408,008,112 | 4,264,006,396 | 0 | 234,000,351 | 26,000,039 | 0 | 7,800,234 | 0 | |
| sphere_intersect | 8,706,013,059 | 9,134,013,701 | 8,494,012,741 | 4,238,006,357 | 0 | 15,600,351 | 9,464,014,196 | 3,016,004,524 | 2,808,004,212 | 0 | 104,000,156 | 26,000,039 | 0 | 10,400,312 | 0 | |
| grid_bounds_intersect | 984,001,476 | 1,004,001,506 | 672,001,008 | 104,000,156 | 0 | 15,600,351 | 962,001,443 | 312,000,468 | 286,000,429 | 0 | 0 | 0 | 0 | 0 | 0 | |
| __kmp_end_split_barrier | 676,001,014 | 624,000,936 | 460,000,690 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| __kmp_x86_pause | 228,000,342 | 224,000,336 | 122,000,183 | 0 | 0 | 10,400,234 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| shader | 216,000,324 | 242,000,363 | 142,000,213 | 104,000,156 | 0 | 0 | 208,000,312 | 104,000,156 | 52,000,078 | 0 | 0 | 0 | 0 | 2,600,078 | 0 | |
| Raypnt | 206,000,309 | 210,000,315 | 208,000,312 | 0 | 0 | 0 | 234,000,351 | 52,000,078 | 78,000,117 | 0 | 0 | 0 | 0 | 0 | 0 | 2,600,03 |
| pos2grid | 204,000,306 | 248,000,372 | 180,000,270 | 26,000,039 | 0 | 0 | 390,000,585 | 26,000,039 | 52,000,078 | 0 | 0 | 0 | 0 | 0 | 0 | |
| tri_intersect | 168,000,252 | 208,000,312 | 180,000,270 | 0 | 0 | 0 | 104,000,156 | 78,000,117 | 52,000,078 | 0 | 52,000,078 | 0 | 0 | 0 | 0 | |
| VScale | 124,000,186 | 126,000,189 | 164,000,246 | 0 | 0 | 0 | 234,000,351 | 52,000,078 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| __kmp_yield | 96,000,144 | 98,000,147 | 200,000,300 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| Selected 1 row(s): | 13,604,020,406 | 14,118,021,177 | 12,572,018,858 | 6,344,009,516 | 0 | 52,001,170 | 14,924,022,386 | 5,408,008,112 | 4,264,006,396 | 0 | 234,000,351 | 26,000,039 | 0 | 7,800,234 | 0 | |

Grouping: Function / Call Stack

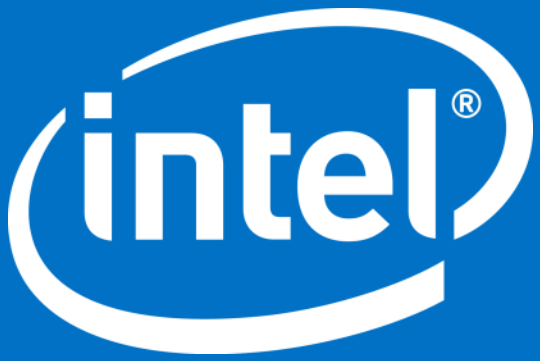
| Function / Call Stack | Clocktic... | Instructions Retired | CPI Rate | MUX Reliability | Filled Pipeline Slots | | Unfilled Pipeline Slots (Stalls) | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Retiring | Bad Speculation | Back-End Bound | Front-end Bound | |
| | | | | | | | | Front-End Latency | Front-End Bandwidth |
| grid_intersect | 14,118,021,177 | 12,572,018,858 | 1.123 | 0.946 | 0.246 | 0.033 | 0.647 | 0.063 | 0.012 |
| sphere_intersect | 9,134,013,701 | 8,494,012,741 | 1.075 | 0.965 | 0.250 | 0.065 | 0.619 | 0.057 | 0.009 |
| grid_bounds_intersect | 1,004,001,506 | 672,001,008 | 1.494 | 0.958 | 0.227 | 0.000 | 0.715 | 0.104 | 0.000 |
| __kmp_end_split_barrier | 624,000,936 | 460,000,690 | 1.357 | 0.000 | 0.000 | 0.000 | 0.792 | 0.167 | 0.042 |
| pos2grid | 248,000,372 | 180,000,270 | 1.378 | 0.636 | 0.367 | 0.000 | 0.633 | 0.000 | 0.131 |
| shader | 242,000,363 | 142,000,213 | 1.704 | 0.860 | 0.322 | 0.000 | 0.946 | 0.000 | 0.027 |
| __kmp_x86_pause | 224,000,336 | 122,000,183 | 1.836 | 0.000 | 0.000 | 0.000 | 0.971 | 0.000 | 0.029 |
| Raypnt | 210,000,315 | 208,000,312 | 1.010 | 0.897 | 0.093 | 0.279 | 0.567 | 0.000 | 0.062 |
| Selected 1 row(s): | 14,118,021,177 | 12,572,018,858 | 1.123 | 0.946 | 0.246 | 0.033 | 0.647 | 0.063 | 0.012 |

# Adding Regression Tests for Performance

- Regression testing isn't just for bugs

1. Create a baseline performance characterization
2. After each change or at a regular interval
    1. Compare new results to baseline
    2. Compare new results to previous results
    3. Evaluate the change
3. goto (1)

- Performance tuning is easier if it's always on your mind and integrated into your development


intel®

# Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.



**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804