

Analyze application performance with Intel® VTune™ Profiler

Roman Khatko



About the speaker



Roman Khatko

- Engineering Manager @ Intel
- Areas of Interest
 - Software Optimization
 - High Performance Computing
 - Developer Tools
- E-mail: roman.p.khatko@intel.com

Agenda

- Software optimization problem
- Baseline
- Software optimization directions
- Summary

Software optimization problem

Optimization Problem

- Define Optimization Criteria
 - Elapsed time
 - Throughput
 - Requests / sec
 - Latency
 - Frames per second
 - Power
 - ...
- Define Stop Criteria
 - Optimization Criteria value when we're done
 - May depend on the tuning potential

Workload

- **System → Workload**
 - Simple case – app itself
 - In complex system – focus on something:
 - Specific data
 - Specific execution scenario
 - ...

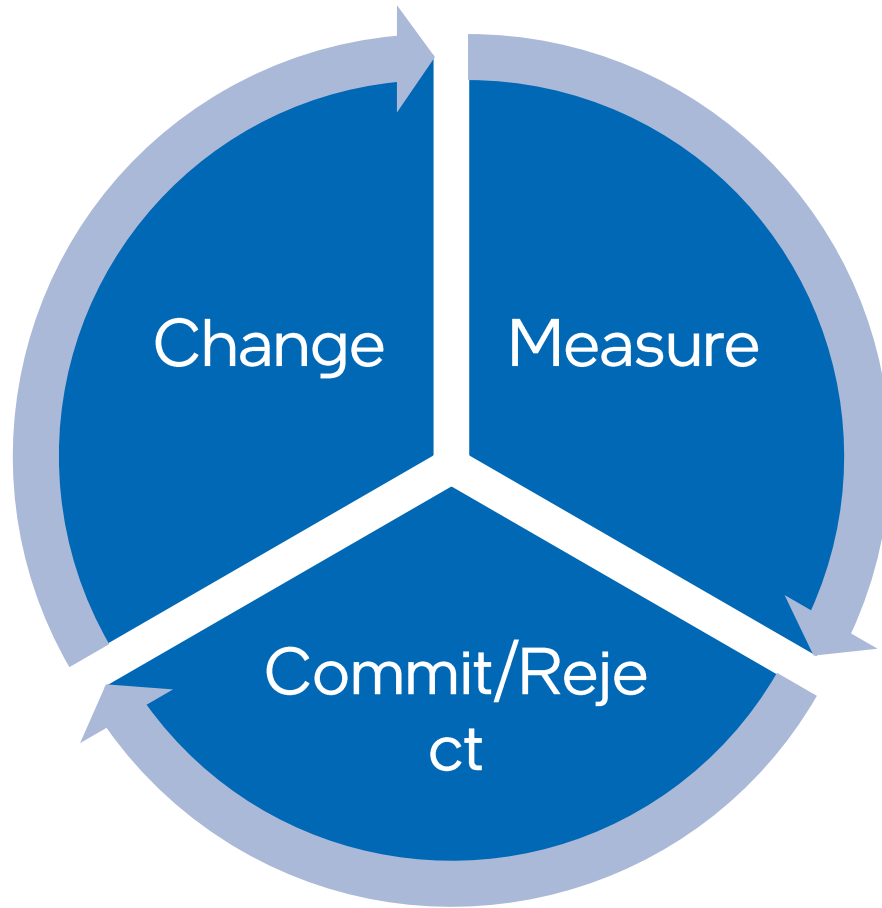
Workload must be:

- **Representative** – Workload improvement leads to system improvement.
- **Measurable** – It is possible to calculate Optimization Criteria after each run.
- **Reproducible** – Optimization Criteria value persists if the workload does not change.

Start with a solid baseline

- Use right compiler flags
 - Compile in release mode
 - Compile with optimization flags (O2 or O3 for gcc, icl, and msvc)
- Compile for the target CPU
 - Your target microarchitecture might have support of more advanced instruction set
 - Benefit from AVX, AVX2, AVX-512
 - -march, -mtune, -mcpu
- Use optimized libraries
 - Intel® oneAPI
- Multiple runs to calculate deviations

Optimization Process



- Start with the baseline
- Iterate
- Finish, when stopping condition is met
- For efficiency, this process should be guided by the profiler

Software optimization directions

Ways to optimize performance and their impact

- Performance increase is unknown and might be huge
 - Algorithmic optimization
 - Design optimization
- Limited performance increase
 - Parallelization
 - Vectorization
 - Memory Access
 - Other microarchitecture optimizations

Optimization level – potential impact

Algorithmic & design optimization



Parallelization & vectorization

Depends..

e.g. N times faster when having N cores – ideally

Microarchitecture optimization

?

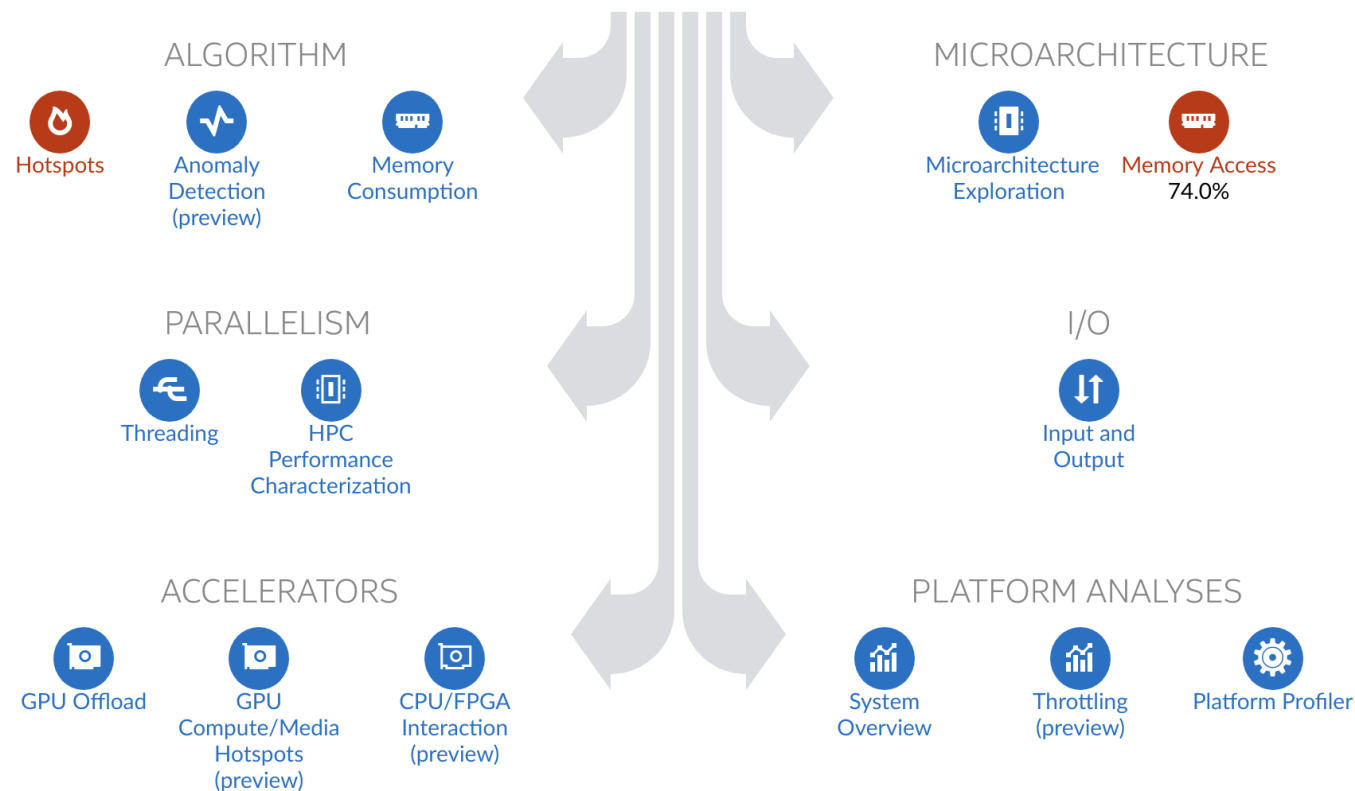
Evaluate the directions – data driven approach

VTune Profiler – Performance Snapshot

- Algorithmic and design optimizations
- Parallelization
- Vectorization
- Offload
- Memory Access
- Other microarchitecture optimizations

Choose your next analysis type

Select a highlighted recommendation based on your performance snapshot.



Algorithmic optimization

VTune Profiler - Hotspots

- Focus on CPU Time
- Identify Hotspots – most time-consuming functions, loops
- Focus on Hotspots
- Choose better algorithms (complexity, constant)
- Use libraries optimized for target HW

Top Hotspots

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

Function	Module	CPU Time [®]
multiply1	matrix	146.389s
init_arr	matrix	0.020s

**N/A is applied to non-summable metrics.*

Software design optimization

VTune Profiler - Hotspots

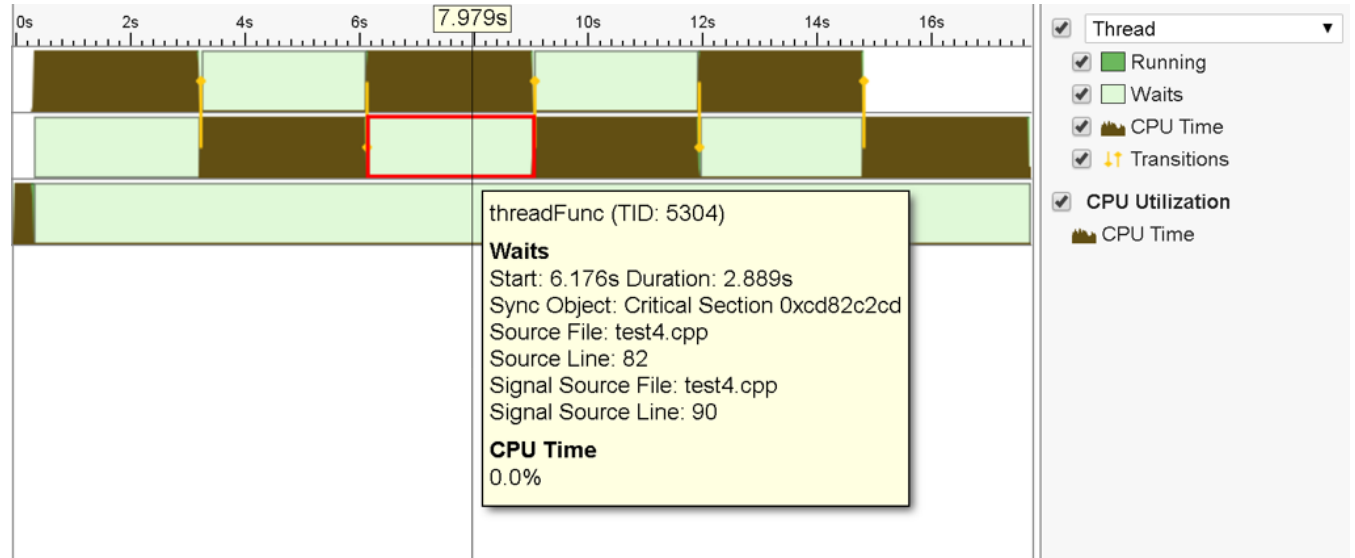
- Algorithm optimization – “calculate fast”
- Design optimization – “avoid calculations”
- Rework application architecture to reduce the calculations
 - Cache frequent requests
 - Avoid unnecessary copies

Parallel optimizations

VTune Profiler - Threading

Perspectives:

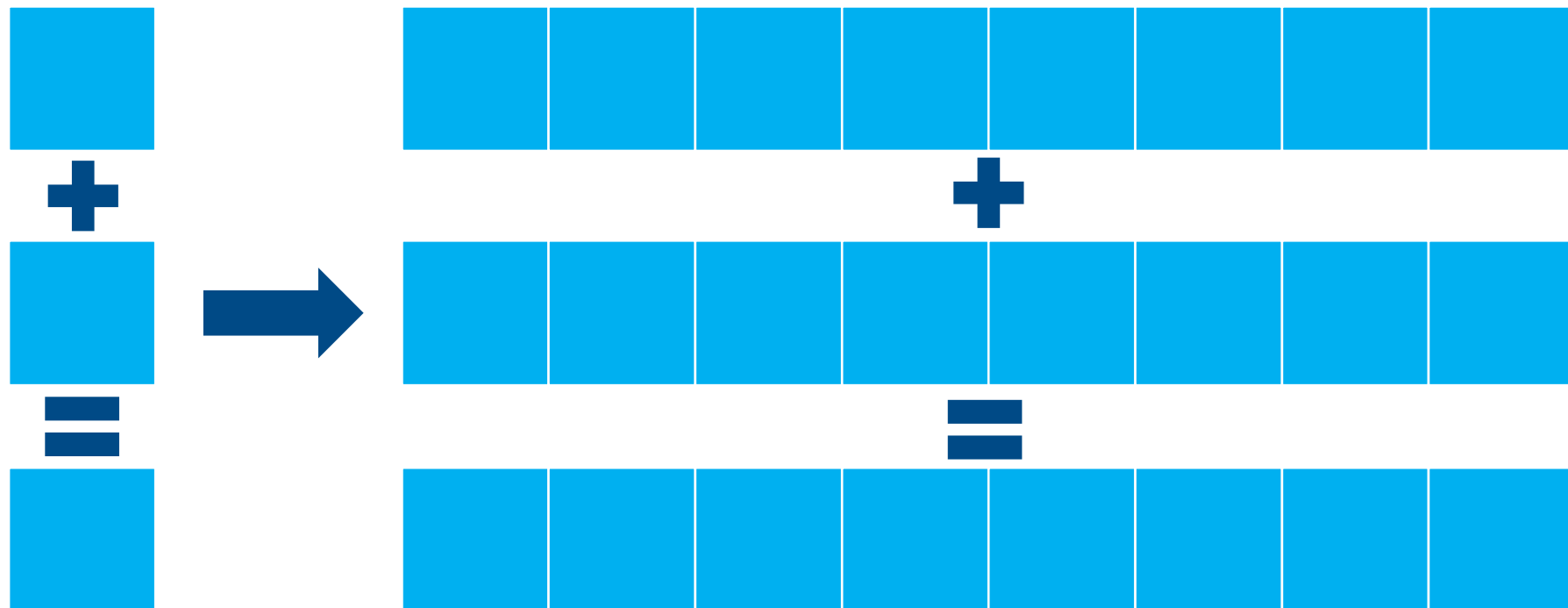
- CPUs
 - CPU Utilization
 - Logical cores, Physical cores
- Threads
 - Effective CPU time, Spin and Overhead Time
 - Inactive time, Sync Wait time, Preemption time
- Sync objects



Vectorization

Intel® Advisor

SIMD instructions:



Vectorization boundaries

AVX-512 zmm register size

int	int	int	int	int	int	int	int	int	int	int	int	int	int	int	int
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

 16x 32-bit int

double	double	double	double	double	double	double	double
--------	--------	--------	--------	--------	--------	--------	--------

 8x 64-bit double

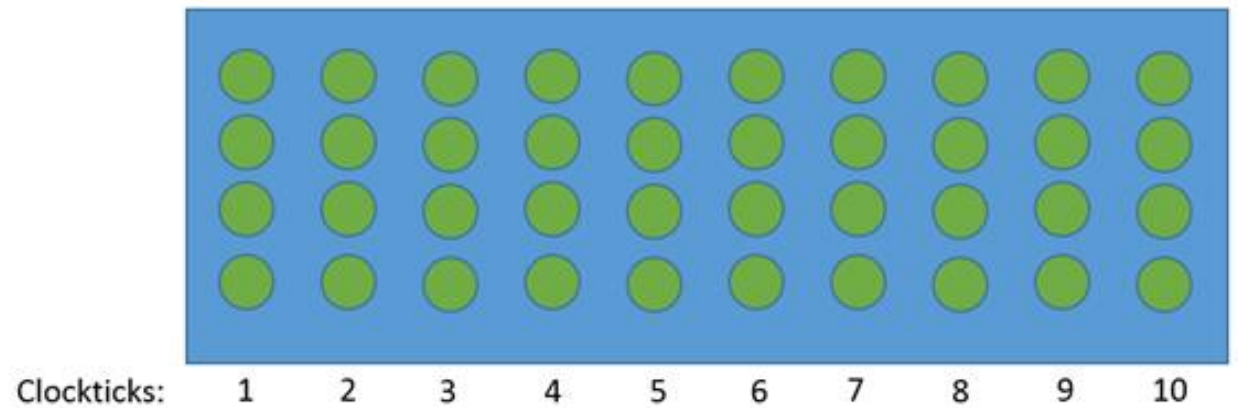
Vectorization

Intel® VTune Profiler – HPC Performance Characterization

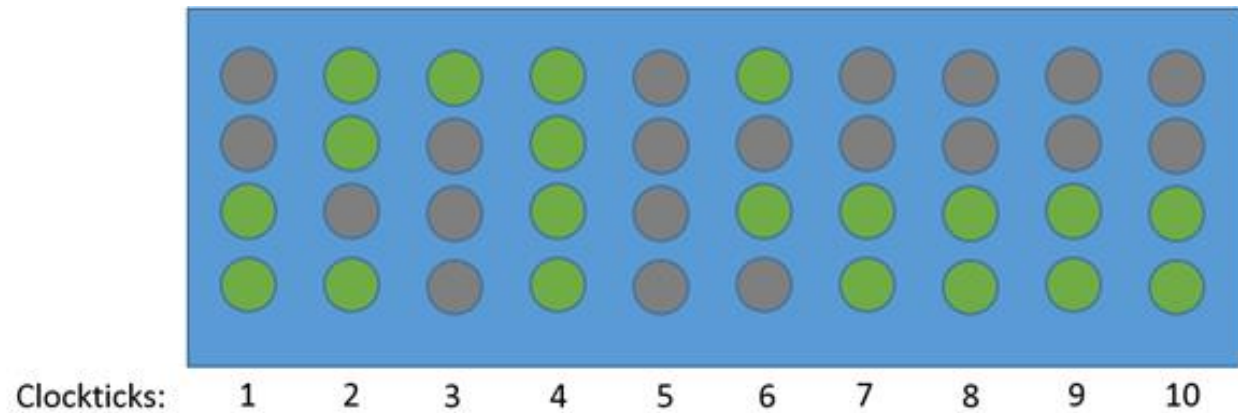
- New CPU's support x86 ISA extensions: AVX, AVX2, AVX-512
- Best to utilize using optimized libraries
 - Update the library when new ISA extension is available
- Pragmas – hints to the compiler
 - Recompile when new ISA extension is available
- Intrinsics & asm
 - Full rewrite when new ISA extension is available
 - Try to avoid, limit to “kernels” otherwise

uArch optimizations

- μ Arch performance gain is limited by ideal CPI (Clocks Per Instruction)
- CPI does not depend on CPU frequency which may change
- Why instructions might not retire:
 - Front-end bound
 - Back-end bound
 - Bad Speculation
- [Top-Down methodology](#)



Ideal CPU pipeline

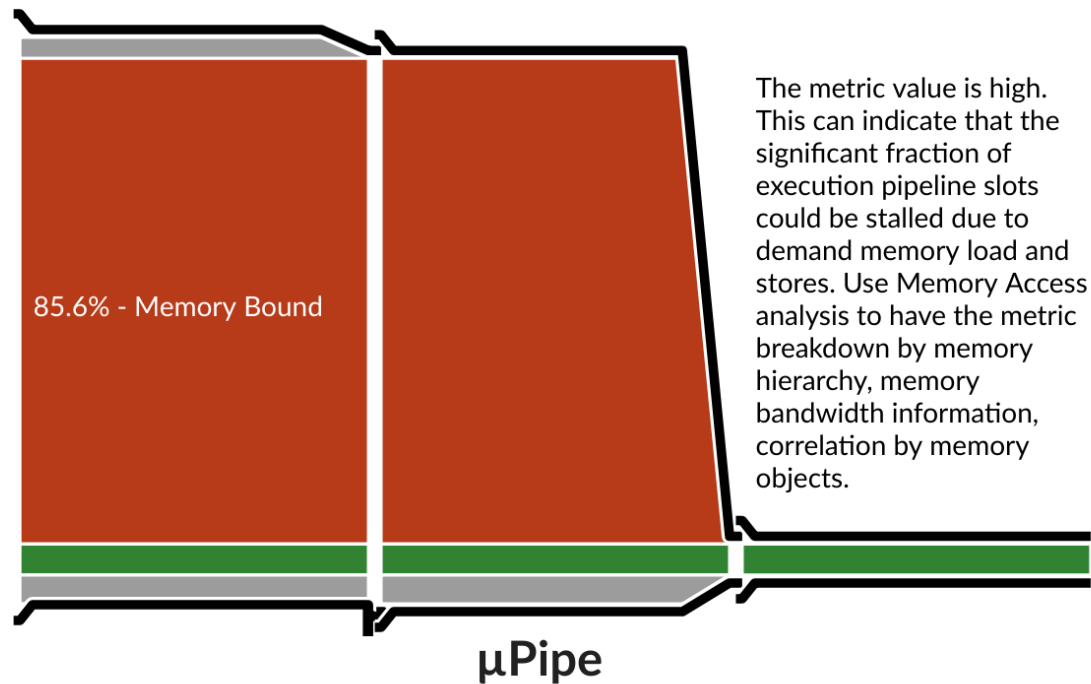


Real CPU pipeline

Microarchitecture optimization – Top down

Elapsed Time[®]: 22.583s

Clockticks:	647,479,000,000	
Instructions Retired:	69,560,600,000	
CPI Rate [®] :	9.308	🚩
MUX Reliability [®] :	0.998	
Retiring [®] :	5.5%	of Pipeline Slots
Front-End Bound [®] :	3.7%	of Pipeline Slots
Bad Speculation [®] :	0.2%	of Pipeline Slots
Back-End Bound [®] :	90.7%	🚩 of Pipeline Slots
Memory Bound [®] :	85.6%	🚩 of Pipeline Slots
L1 Bound [®] :	0.0%	of Clockticks
L2 Bound [®] :	0.0%	of Clockticks
L3 Bound [®] :	0.8%	of Clockticks
DRAM Bound [®] :	89.4%	🚩 of Clockticks
Store Bound [®] :	0.0%	of Clockticks
Core Bound [®] :	5.1%	of Pipeline Slots
Average CPU Frequency [®] :	3.7 GHz	
Total Thread Count:	9	
Paused Time [®] :	0s	



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 shape gets more narrow.

Top-down metrics can be applied to the specific hotspot

Memory Access optimizations

VTune Profiler – Memory Access

- Start with Top-Down methodology to locate the problem
- Make your app NUMA-aware
- Reduce frequent DRAM accesses
- Fix false sharing issues
- Add hints for prefetchers

Summary

- Use optimized libraries
- Get a solid baseline
- Define optimization criteria for your workload
- Profile your code
- Optimize your code design and algorithms first
- Vectorize and parallelize
- Offload to the accelerators
- Optimize memory access
- Help the compiler rather than play with assembly code

Links

- [Intel® VTune™ Profiler](#)
- [Intel® VTune™ Profiler Performance Analysis Cookbook](#)
- [Intel® oneAPI Toolkits](#)
- [Intel® Advisor](#)

Q&A

The Intel logo is centered on a solid blue background. It features the word "intel" in a white, lowercase, sans-serif typeface. A small, bright blue square is positioned above the first vertical stroke of the letter 'i'. To the right of the word "intel" is a small white registered trademark symbol (®).

intel®