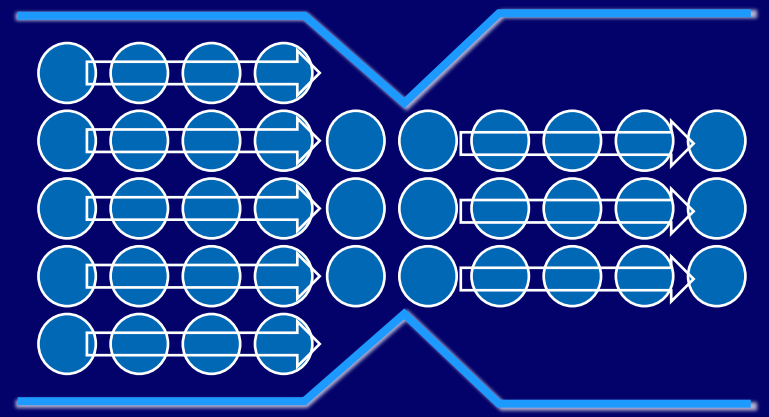

intel
Data Center and AI Group
Swanand Mhalagi
Harshad Sane
Kshitij Doshi

Workload optimization challenges, General

A few of the common ones out of many

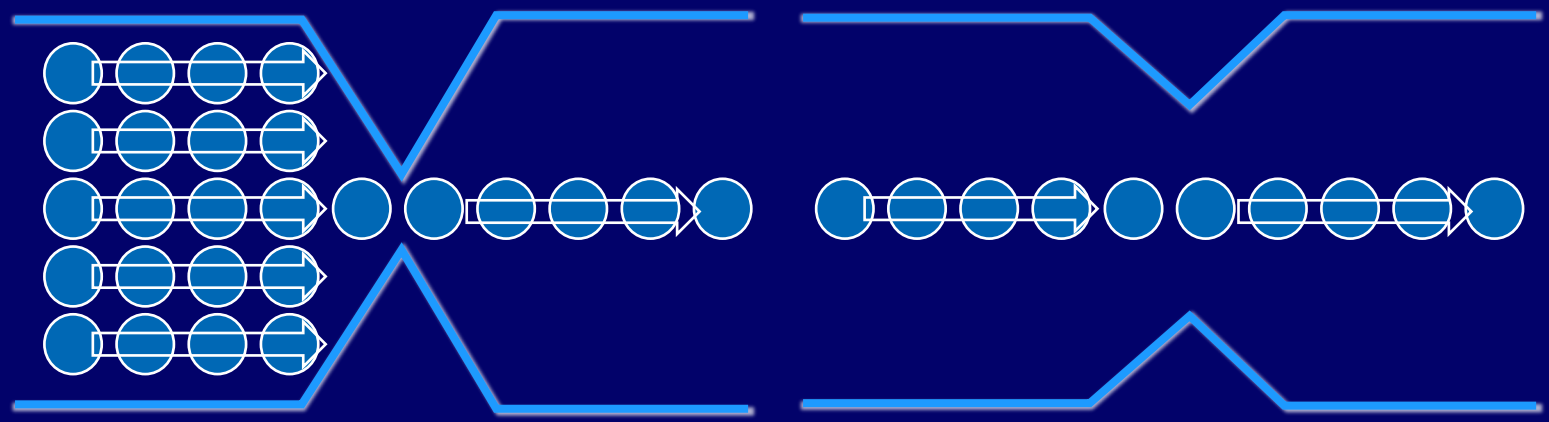
- Identifying software and hardware bottlenecks
- Separating limiting cases from inefficient usage of resources
- Improper parameters and/or incorrect platform configurations



Workload optimization challenges, General

A few of the common ones out of many

- Identifying software and hardware bottlenecks
- Separating limiting cases from inefficient usage of resources
- Improper parameters and/or incorrect platform configurations

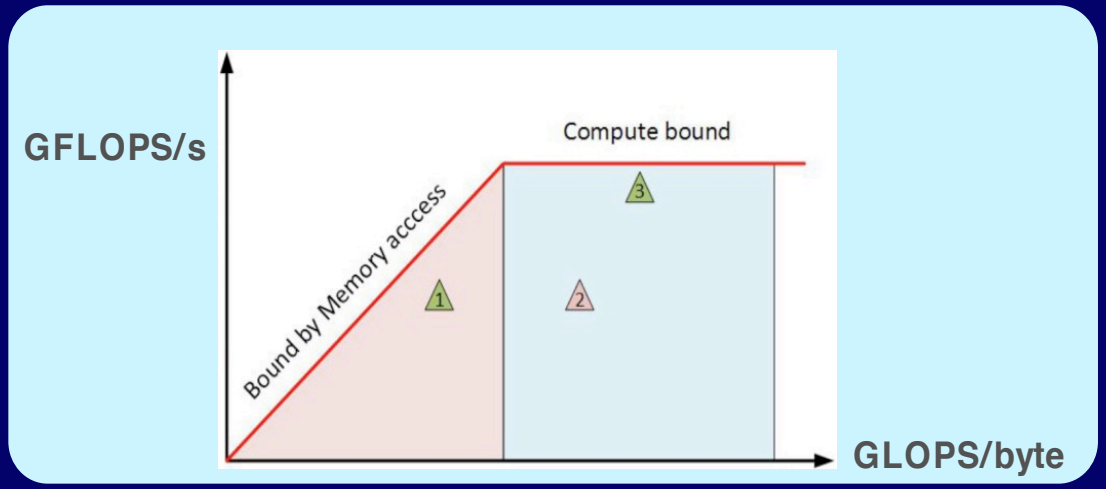


Workload optimization challenges, General

A few of the common ones out of many

- Identifying software and hardware bottlenecks
- Separating limiting cases from inefficient usage of resources
- Improper parameters and/or incorrect platform configurations

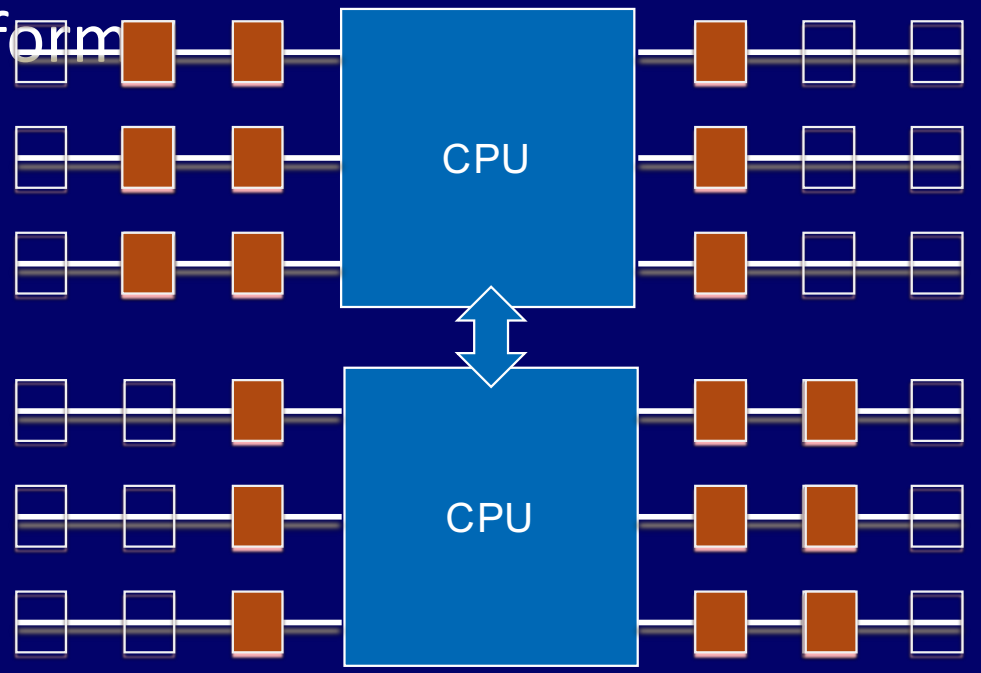
https://www.intel.com/content/dam/develop/public/us/en/documents/roofline-analysis-with-intel-advisor.pdf



Workload optimization challenges, General

A few of the common ones out of many

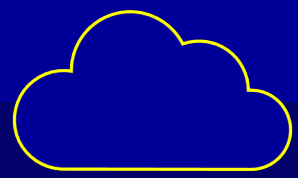
- Identifying software and hardware bottlenecks
- Separating limiting cases from inefficient usage of resources
- Improper parameters and/or incorrect platform configurations

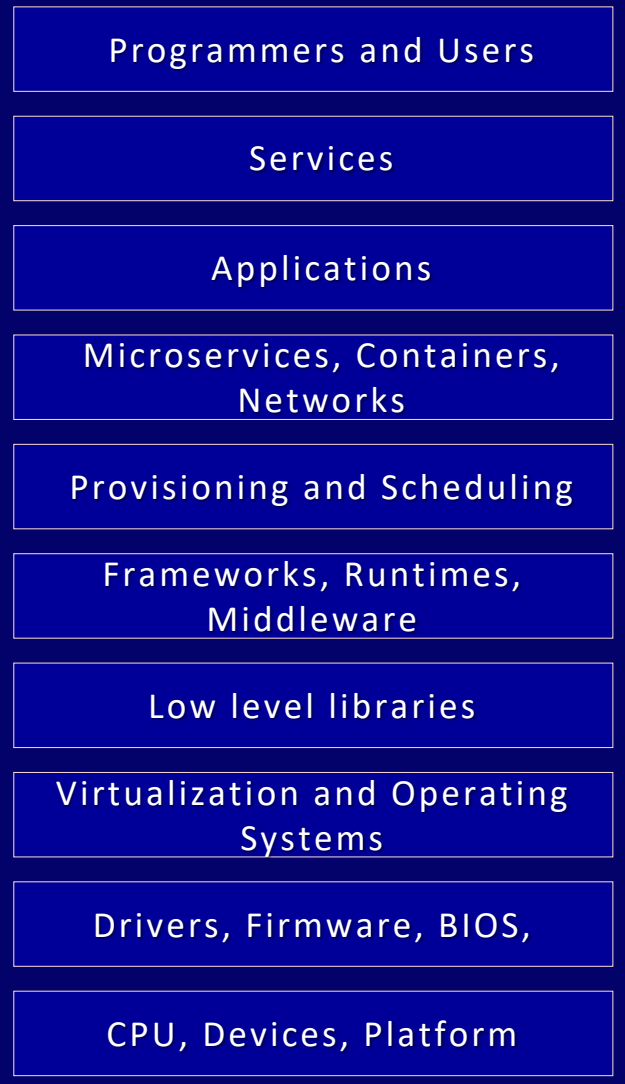


Workload optimization challenges in the cloud

Again, a few of the common ones out of many

- Abstraction layers
- Observability
- Control
- Productivity vs performance programming
- Tail latency (throughput is often secondary)

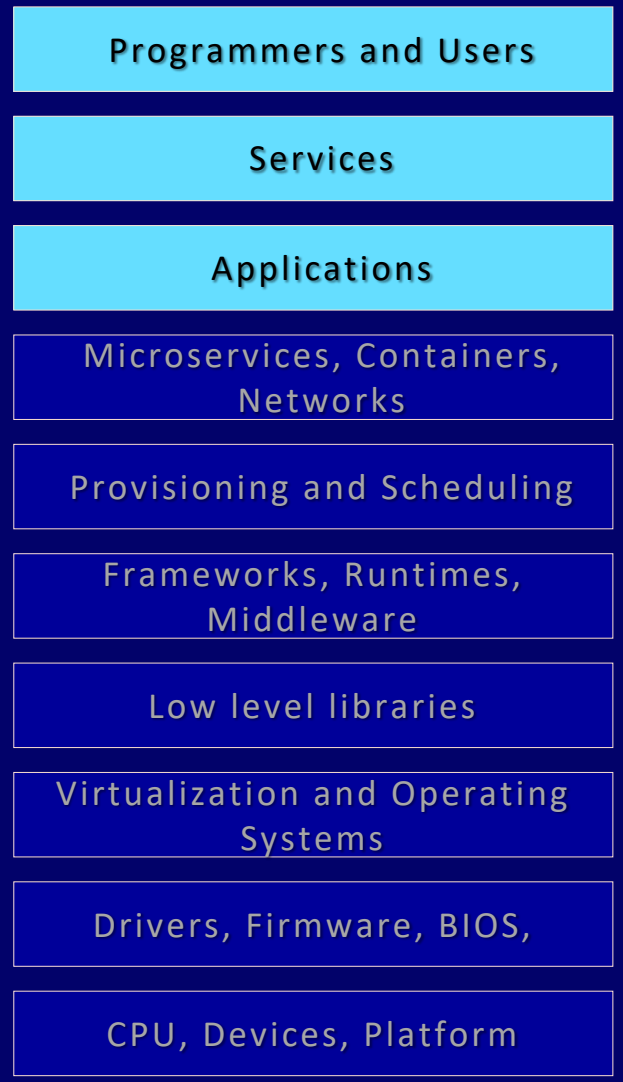




Workload optimization challenges in the cloud

Again, a few of the common ones out of many

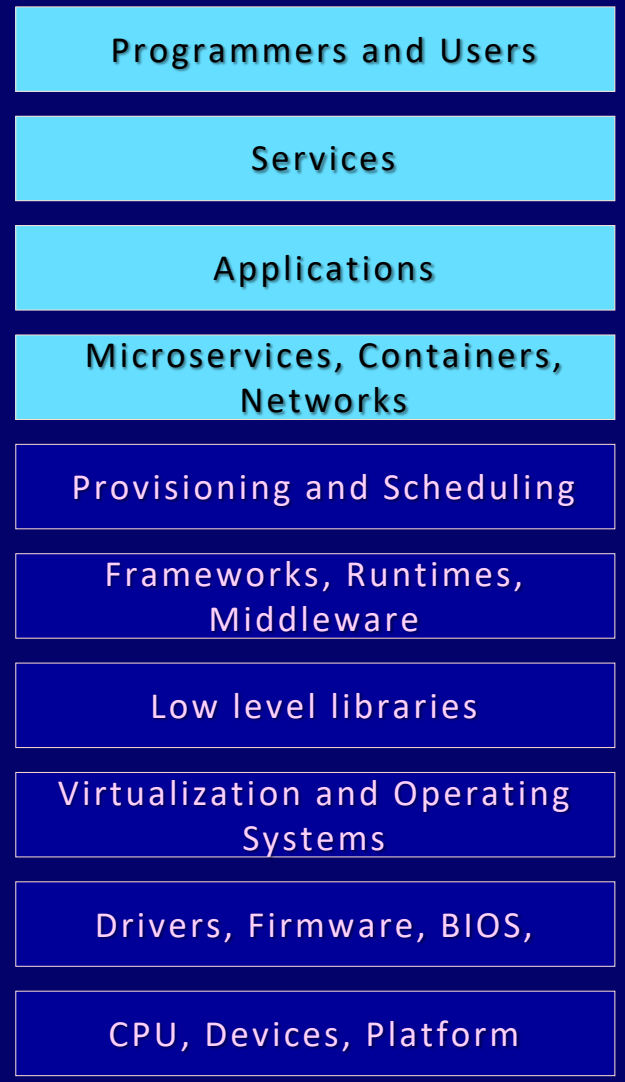
- Abstraction layers
- Observability
- Control
- Productivity vs performance programming
- Tail latency (throughput is often secondary)



Workload optimization challenges in the cloud

Again, a few of the common ones out of many

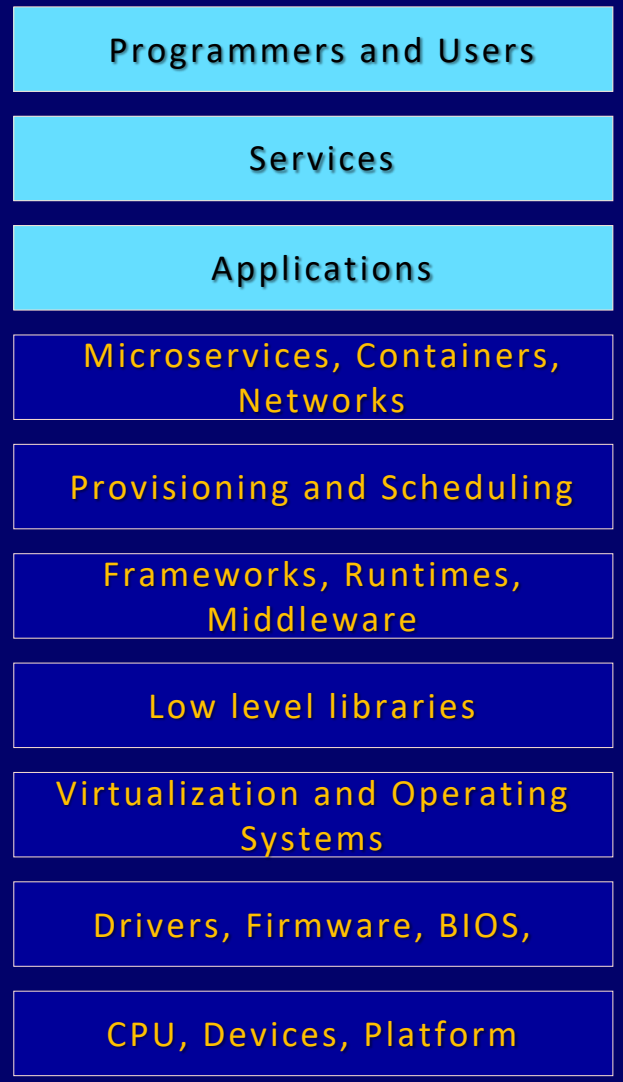
- Abstraction layers
- Observability
- Control
- Productivity vs performance programming
- Tail latency (throughput is often secondary)



Workload optimization challenges in the cloud

Again, a few of the common ones out of many

- Abstraction layers
- Observability
- Control
- Productivity vs performance programming
- Tail latency (throughput is often secondary)



Accelerate libraries with Intel® Distribution for Python*

High Performance Python* for Scientific Computing, Data Analytics, Machine Learning

FASTER PERFORMANCE	GREATER PRODUCTIVITY	ECOSYSTEM COMPATIBILITY
Performance Libraries, Parallelism, Multithreading, Language Extensions	**Prebuilt & Accelerated Packages**	**Supports Python* 2.7 & 3.6, & 3.7 conda, pip**
Accelerated NumPy*/SciPy*/scikit-learn* with oneMKL[1] & oneDAL[2]	Prebuilt & optimized packages for numerical computing, machine/deep learning, HPC & data analytics	Compatible & powered by Anaconda*, supports conda & pip
Data analytics, machine learning with scikit-learn, daal4py	**Drop-in replacement for existing Python*** **Usually NO code changes required!**	Distribution & individual optimized **oneMKL accelerated NumPy*, and SciPy now in Anaconda*!**
Optimized run-times Intel MPI®, Intel® TBB	Conda build recipes included in packages	Optimizations upstreamed to main Python* trunk
Scale with Numba* & Cython*	Free download & free for all uses including commercial deployment	Commercial support through Intel® Parallel Studio XE
Includes optimized mpi4py, works with Dask* & PySpark*		
Optimized for latest Intel® architecture		

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS[1]*

[1]Intel® oneAPI Math Kernel Library
[2]Intel® oneAPI Data Analytics Library

intel

Workload optimization challenges in the cloud

Again, a few of the common ones out of many

- Abstraction layers
- Observability
- Control
- Productivity vs performance programming
- Tail latency (throughput is often secondary)

Challenges in financial/technical domain

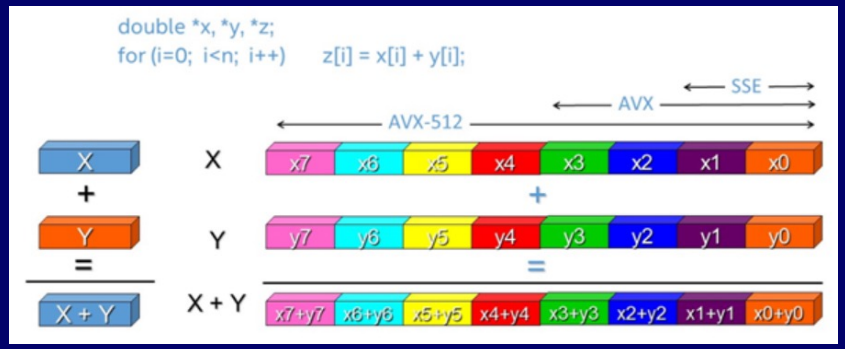
Again, a few of the common ones out of many

- High sensitivity to hardware (freq, cache and mem BW, cpu & socket affinity, NUMA)
- Efficient use of vector instructions
- Efficient use of threading
- Memory management strategies across applications, libraries, operating system

Challenges in financial/technical domain

Again, a few of the common ones out of many

- High sensitivity to hardware (freq, cache and mem BW, cpu & socket affinity, NUMA)
- Efficient use of vector instructions
- Efficient use of threading
- Memory management strategies across applications, libraries, operating system



Source: ***Vectorization Opportunities for Improved Performance with Intel® AVX-512***
https://www.codeproject.com/Articles/1182515/Vectorization-Opportunities-for-Improved-Perform

Challenges in financial/technical domain

Again, a few of the common ones out of many

- High sensitivity to hardware (freq, cache and mem BW, cpu & socket affinity, NUMA)
- Efficient use of vector instructions
- Efficient use of threading
- Memory management strategies across applications, libraries, operating system

Challenges in financial/technical domain

Again, a few of the common ones out of many

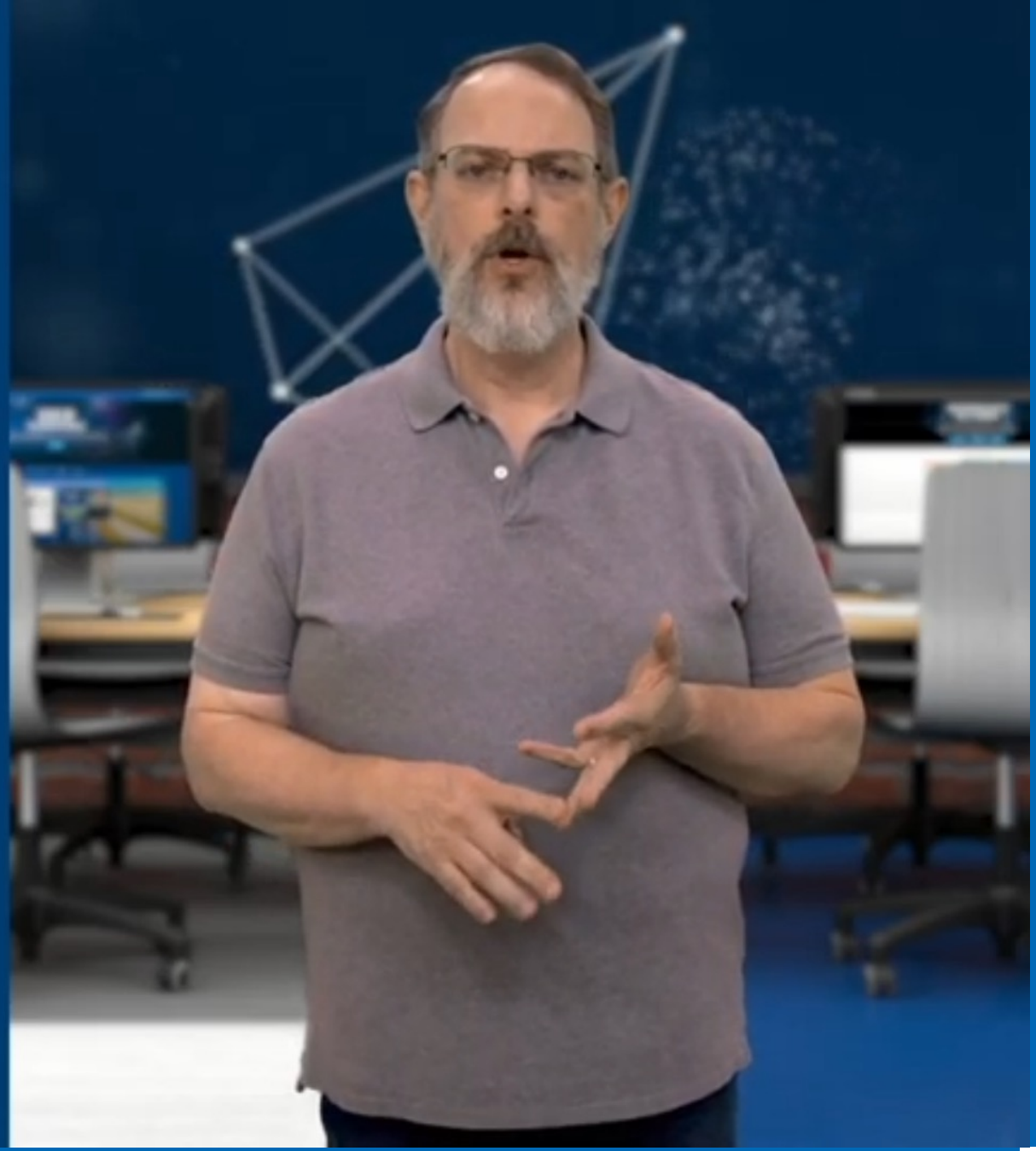
- High sensitivity to hardware (freq, cache and mem BW, cpu & socket affinity, NUMA)
- Efficient use of vector instructions
- Efficient use of threading
- Memory management strategies across applications, libraries, operating system

https://www.youtube.com/watch?v=VmO8gN1VgME

INTEL® DISTRIBUTION FOR PYTHON

Includes accelerated versions of packages such as:

- NumPy
- SciPy
- SciKit Learn

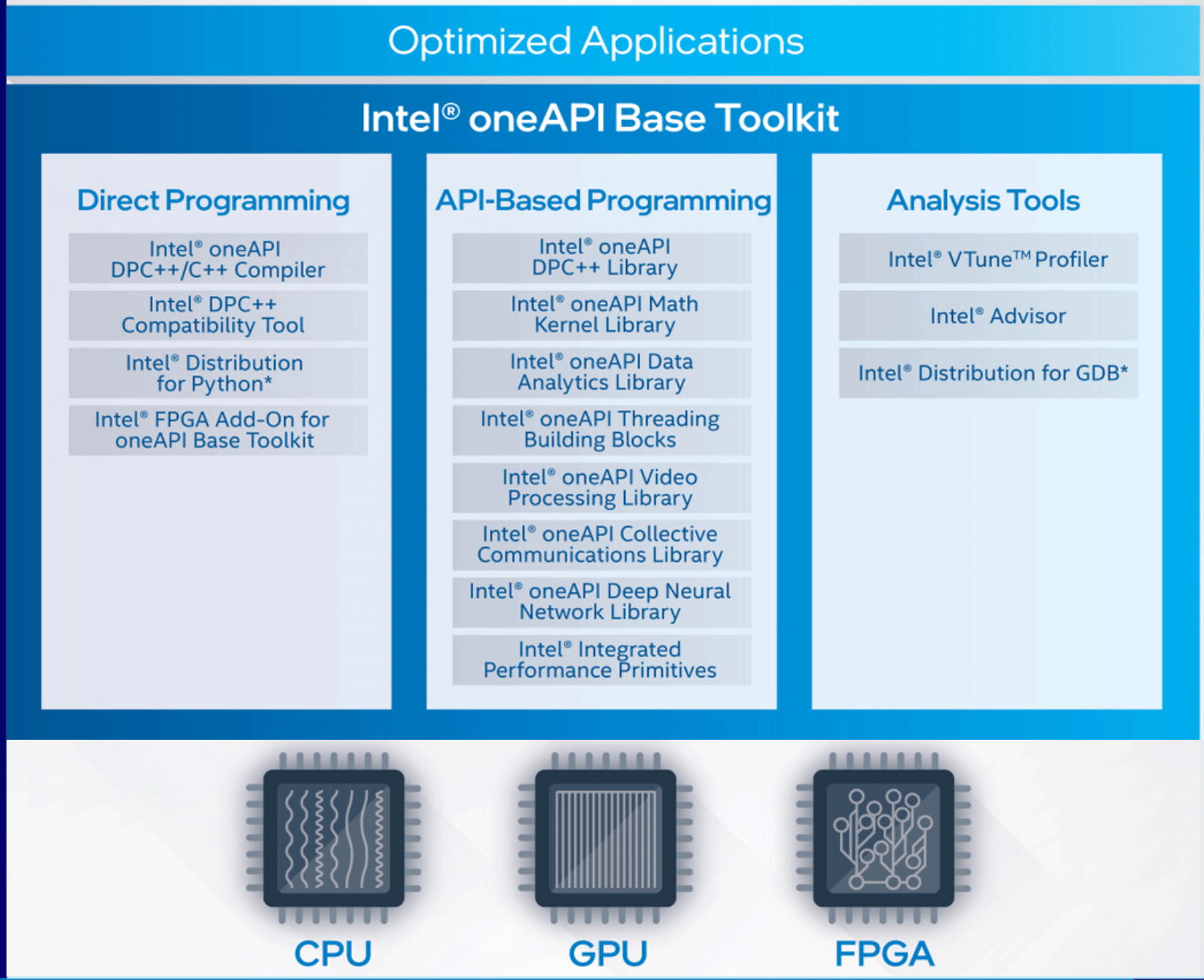


intel

Intel® Contributes AVX-512 Optimizations To Numpy, Yields Massive Speedups

"…provides optimized versions of … the major math functions… in both single and double precision modes. … Intel engineers found that even with older Intel Skylake X processors this meant Numpy was running up to 55x faster in select functions. The average speed-up was 14x for double precision and 32x for single precision performance."

https://www.phoronix.com/scan.php?page=news_item&px=Intel-Numpy-AVX-512-Landed

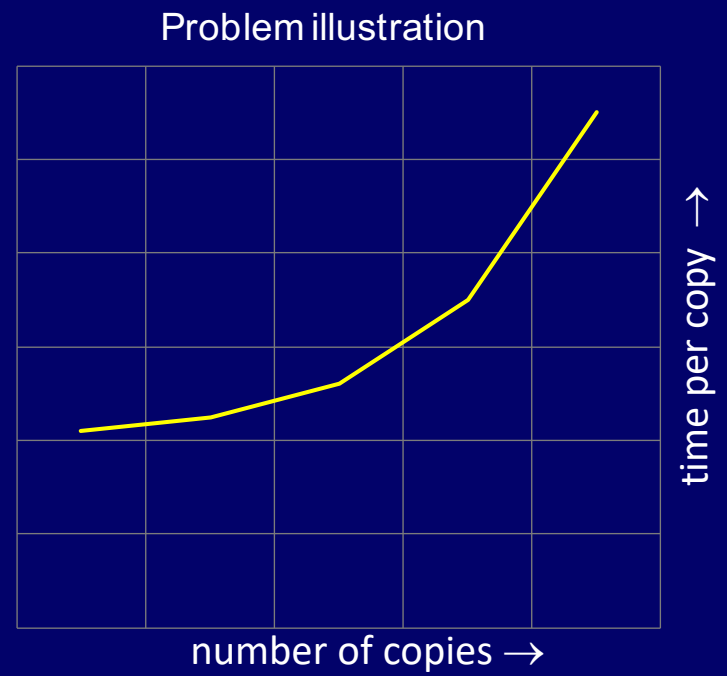


A recent journey through helping a customer with their performance puzzle

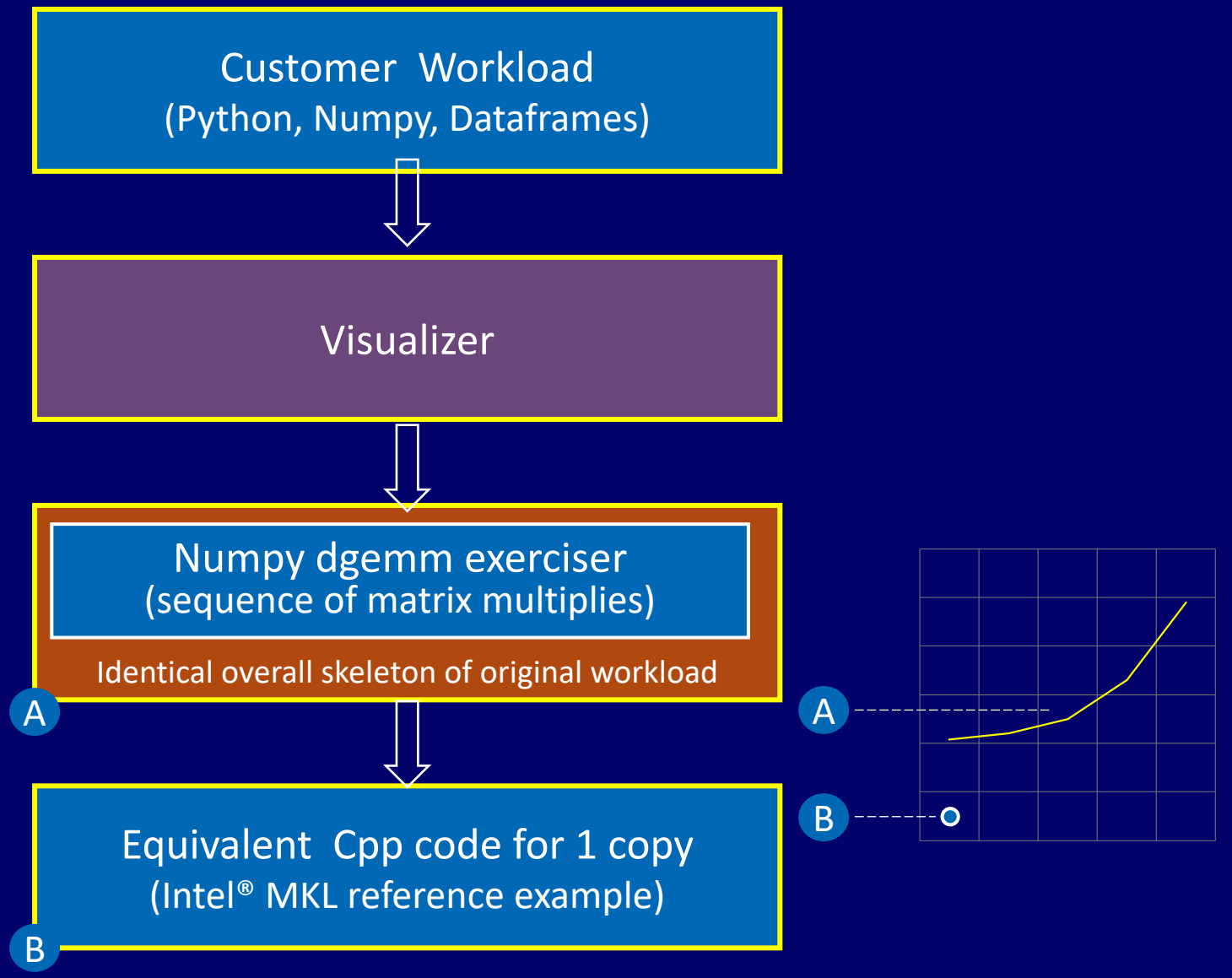
Customer example problem

Extremely poor scaling with numbers of instances of the solution

- $\mathcal{N}$ copies on same cloud node
- No data or control dependencies
- Time $\propto$ faster-than $\mathcal{N}$
- Time at $\mathcal{N}=1$ also high

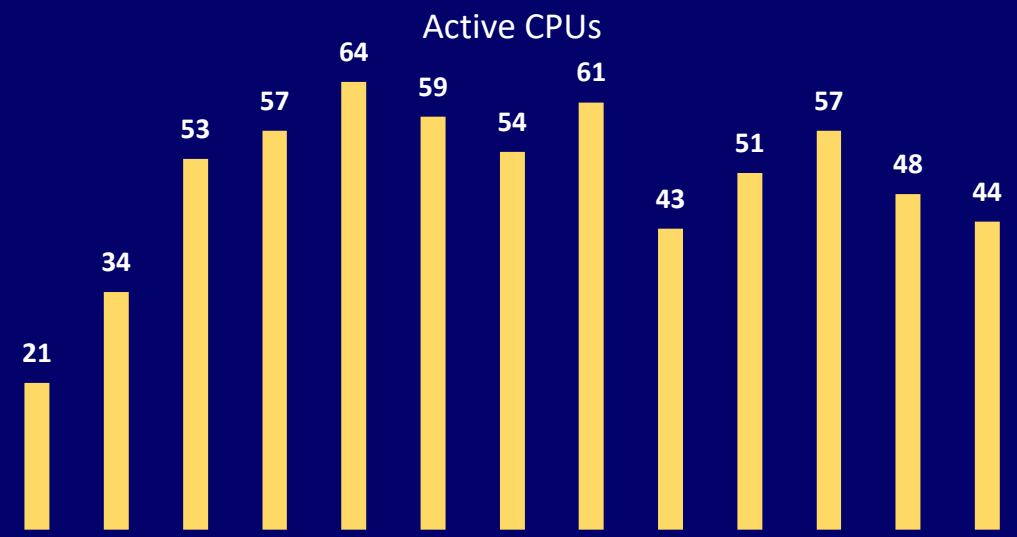


Simplifying proxy

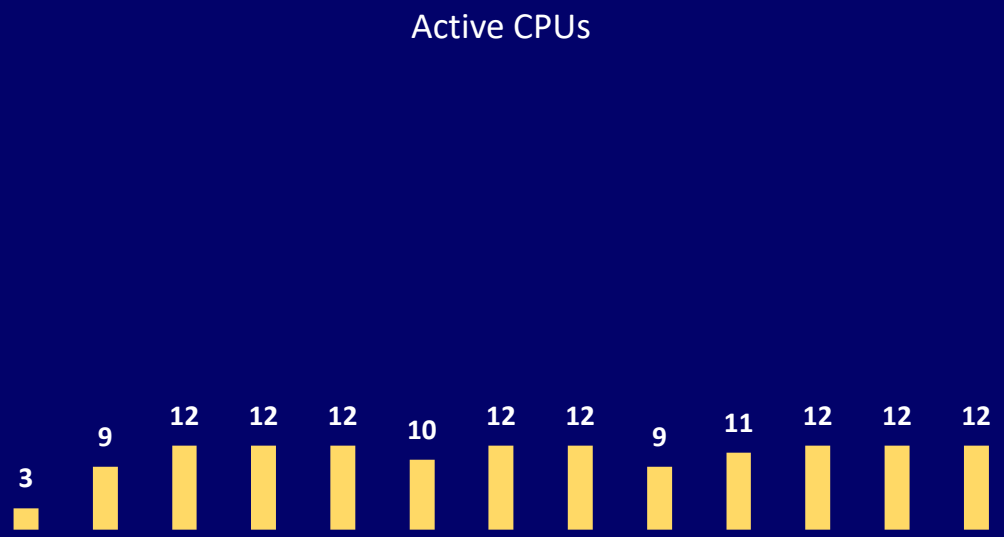


Thread oversubscription

For one copy, dgemm numpy exerciser



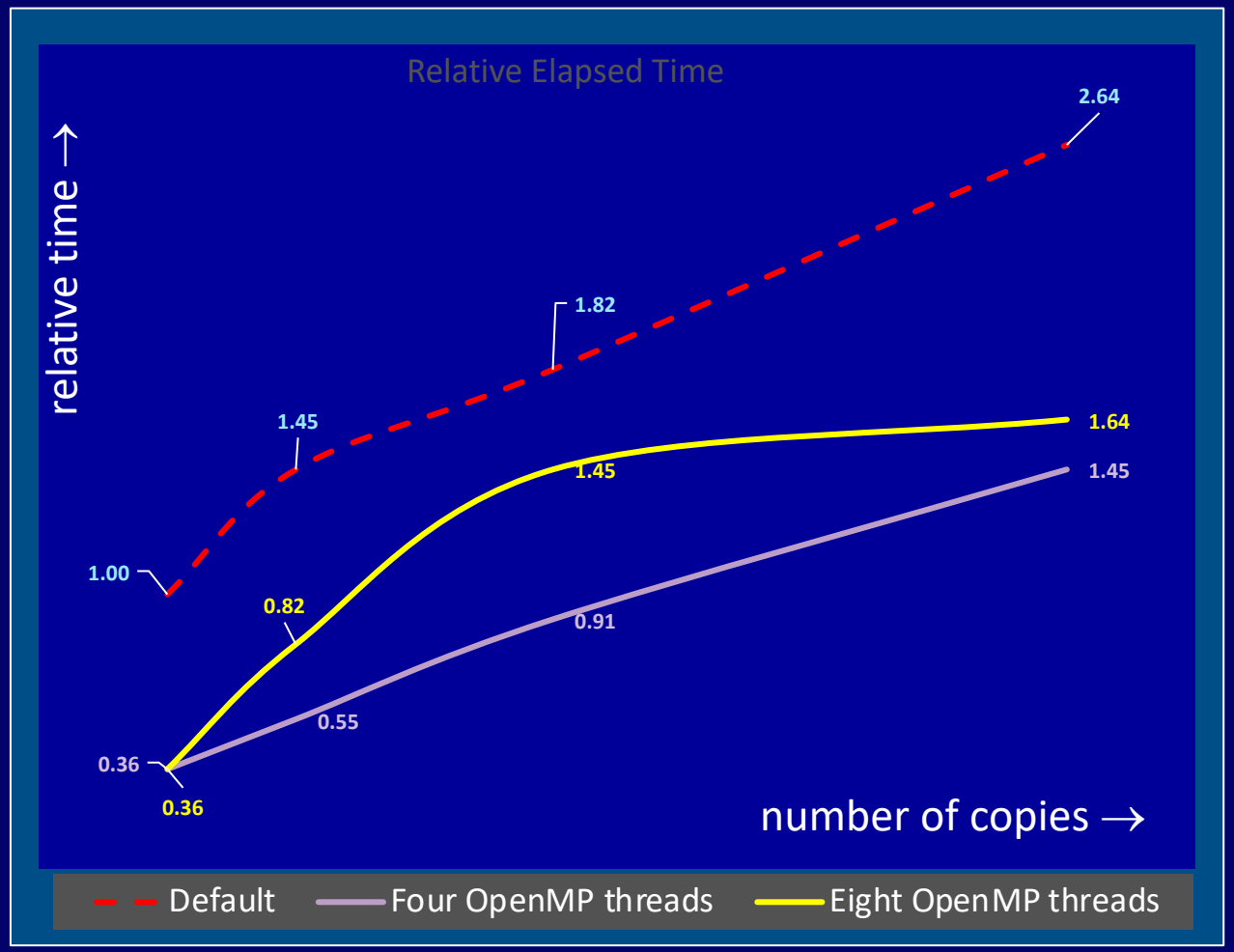
For one copy, dgemm Cpp exerciser



What was happening: Open MP settings getting reset across forks

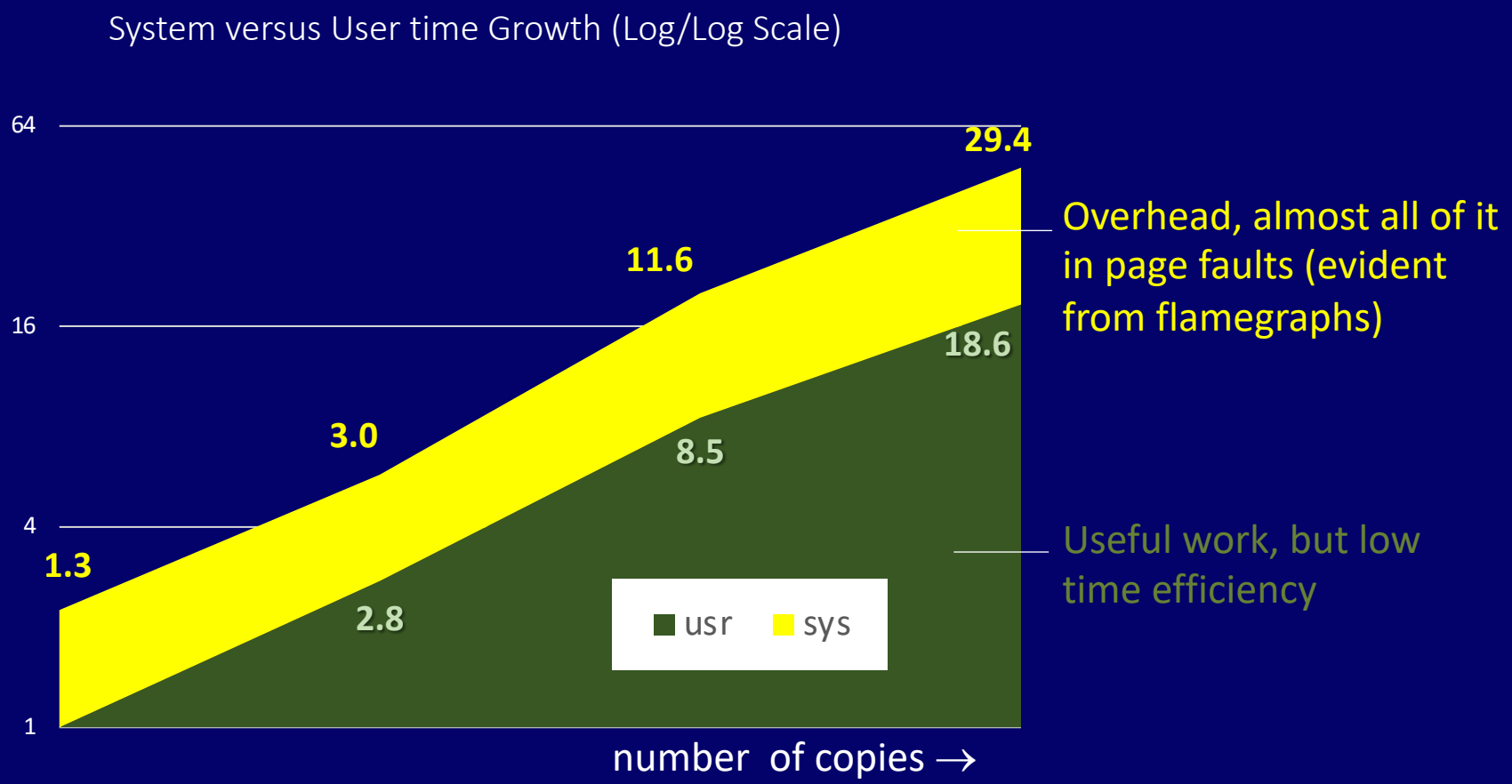
Correcting for thread oversubscription

with OMP_NUM_THREADS setting



Roughly 2X-3X improvement in elapsed time by preventing over-activity

How the time was being spent



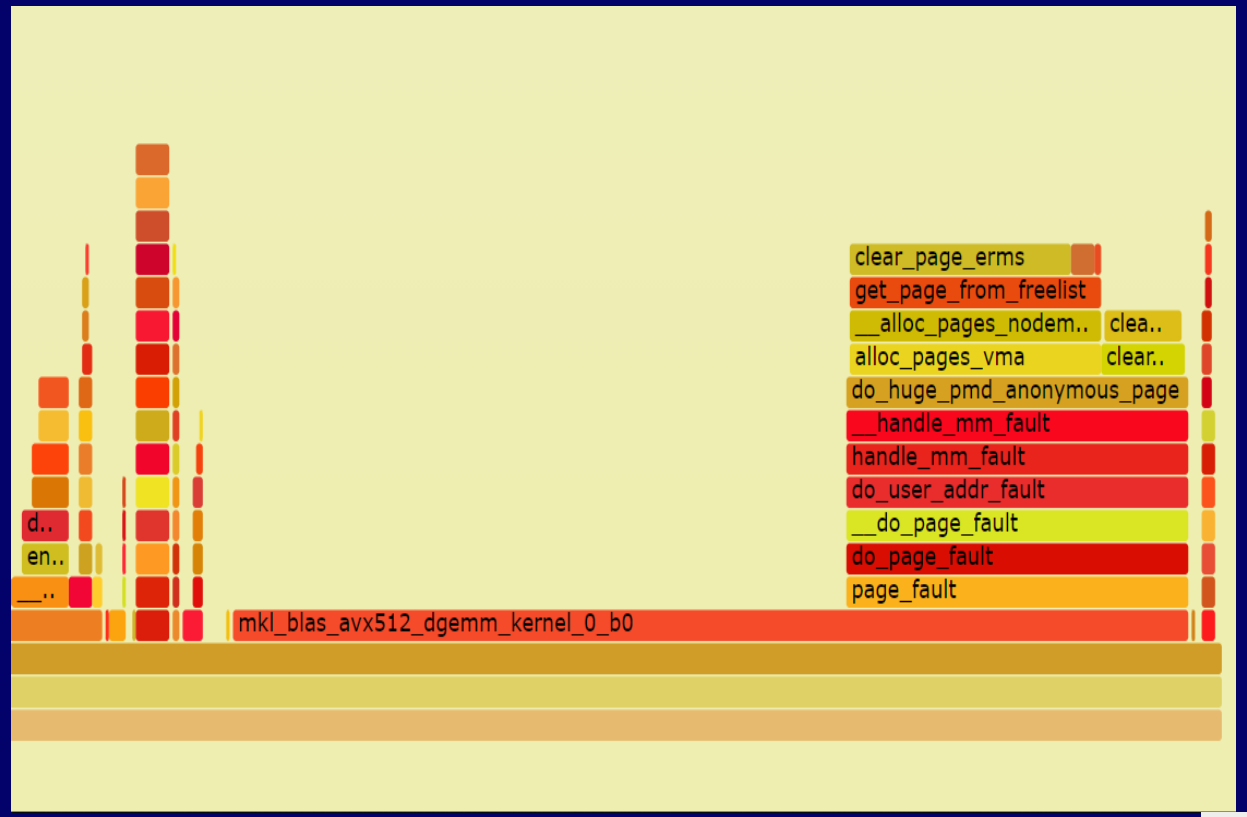
Overhead, almost all of it in page faults (evident from flamegraphs)

Useful work, but low time efficiency

By comparison, flamegraphs for Cpp reference code showed almost all time in usr, and, high time efficiency.

With four OpenMP threads

Where were the processor cycles going?



Numpy

Flame graphs facilitate visualization of how time builds up across call hierarchies

https://www.brendangregg.com/flamegraphs.html

C++

Correction

```
1  #!/usr/bin/env python3
2
3  import os
4  import numpy as np
5          ...
8  os.environ['MKL_NUM_THREADS'] = str('6')
9  os.environ['MKL_DYNAMIC'] = str('0')
10 os.environ['OMP_NUM_THREADS'] = str('8')
```

```
#!/usr/bin/env python3

import os
        ...
os.environ['MKL_NUM_THREADS'] = str('6')
os.environ['MKL_DYNAMIC'] = str('0')
os.environ['OMP_NUM_THREADS'] = str('8')
        ...
import numpy as np
```

mkl-service and MKL_VERBOSE

…

…

mkl-service + Intel(R) MKL: THREADING LAYER: (null)

mkl-service + Intel(R) MKL: setting Intel(R) MKL to use INTEL OpenMP runtime

mkl-service + Intel(R) MKL: preloading libiomp5.so runtime

MKL_VERBOSE oneMKL 2021.0 Update 3 Product build 20210617 for Intel(R) 64 architecture Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512) with support of Intel(R) Deep Learning Boost (Intel(R) DL Boost), Lnx 1.00GHz lp64 intel_thread

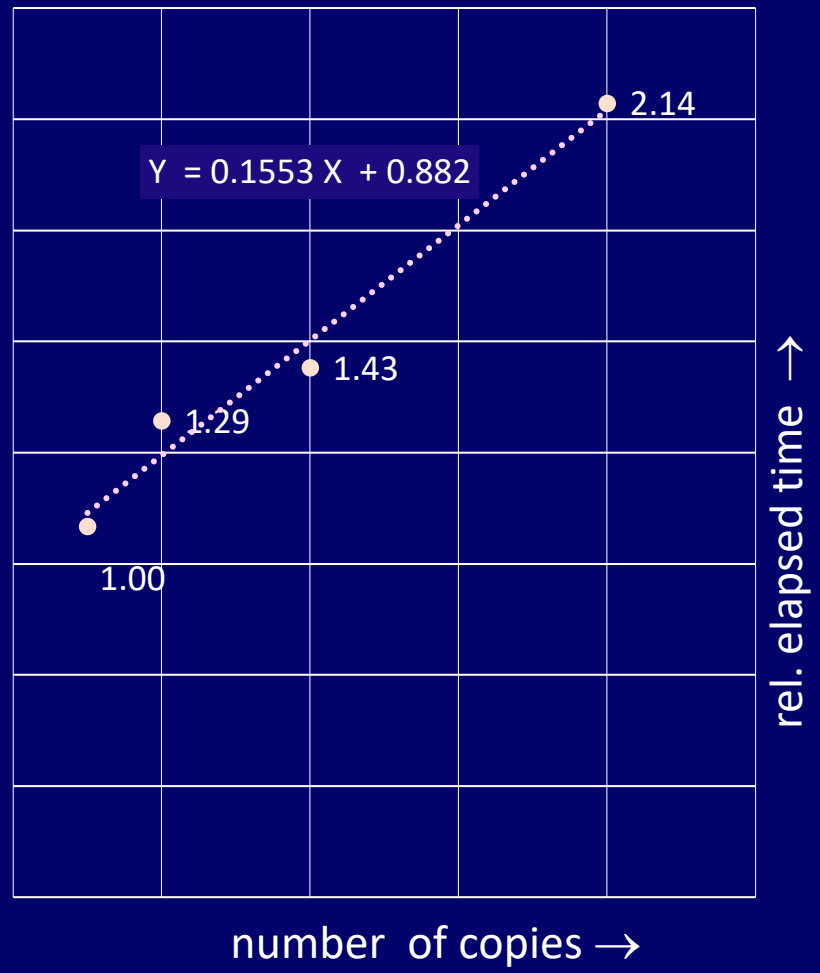
…

…

…

Performance resource: Developer guide for MKL
https://www.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/managing-performance-and-memory/improving-performance-with-threading/avoiding-conflicts-in-the-execution-environment.html

Result



With 6 OpenMP threads per copy

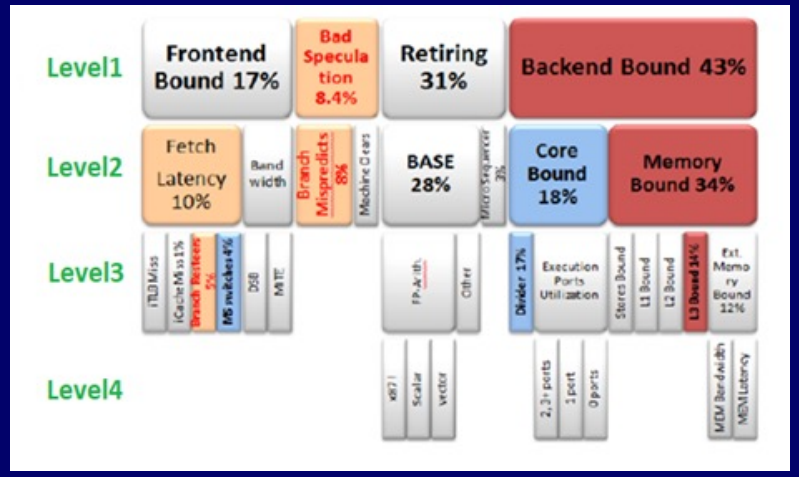
We still get a linear increase in time, but with a small coefficient (0.16) per additional unit of work

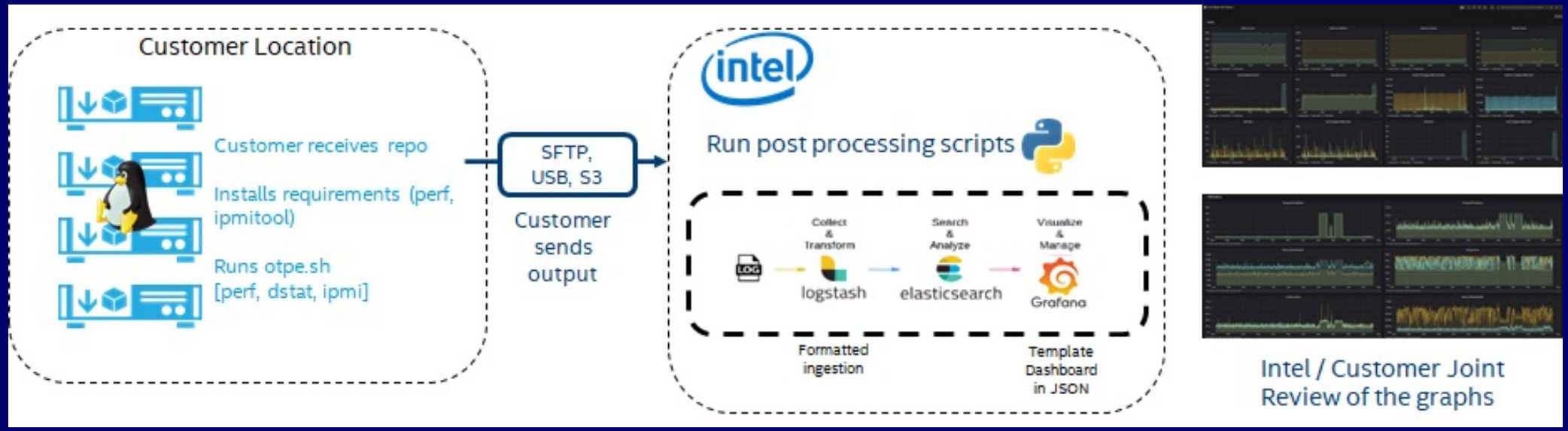
Performance tools which are already public or being made public . . .

Perfspect and Intel® Telemetry Collector (ITC)

Cloud-ready lightweight hardware event monitor

- PerfSpect (https://github.com/intel/PerfSpect) provides a hierarchical breakdown of cpu resources
- ITC utilizes PerfSpect along with other Linux utilities to provide a sclable collection and visualization platform telemetry

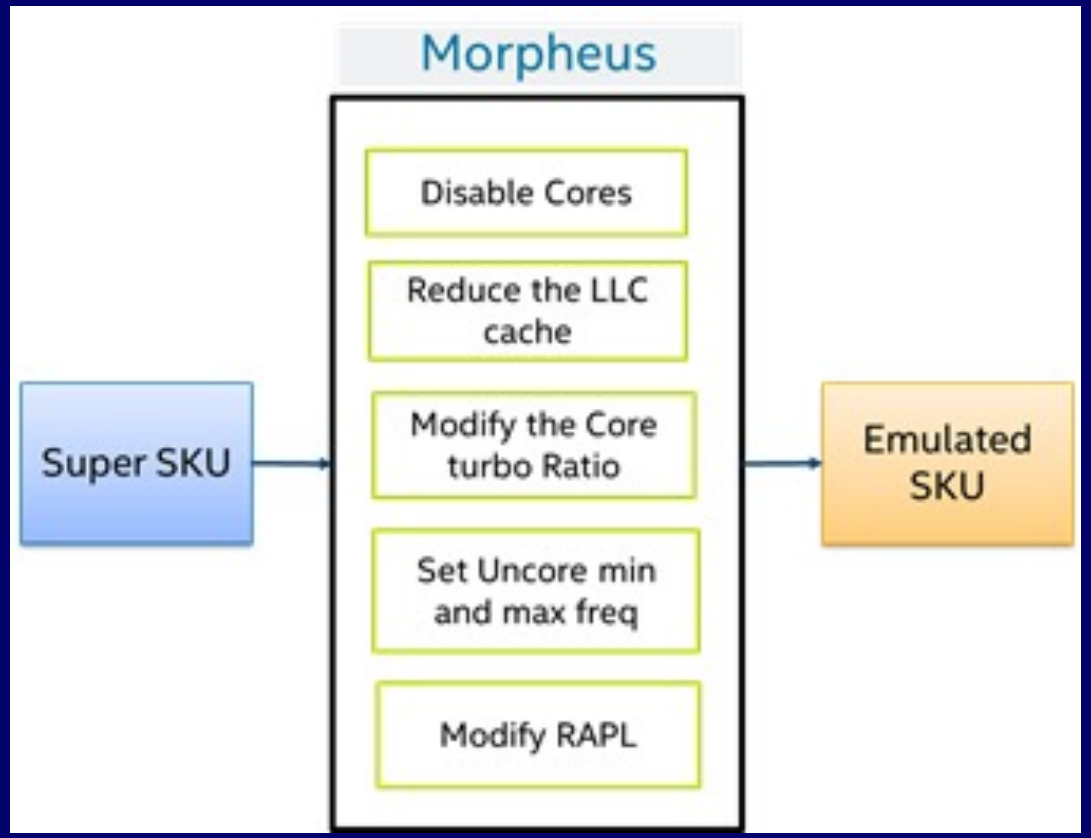




Intel® SKU emulator

On-prem and hybrid cloud exploration with software-controlled SKU emulation

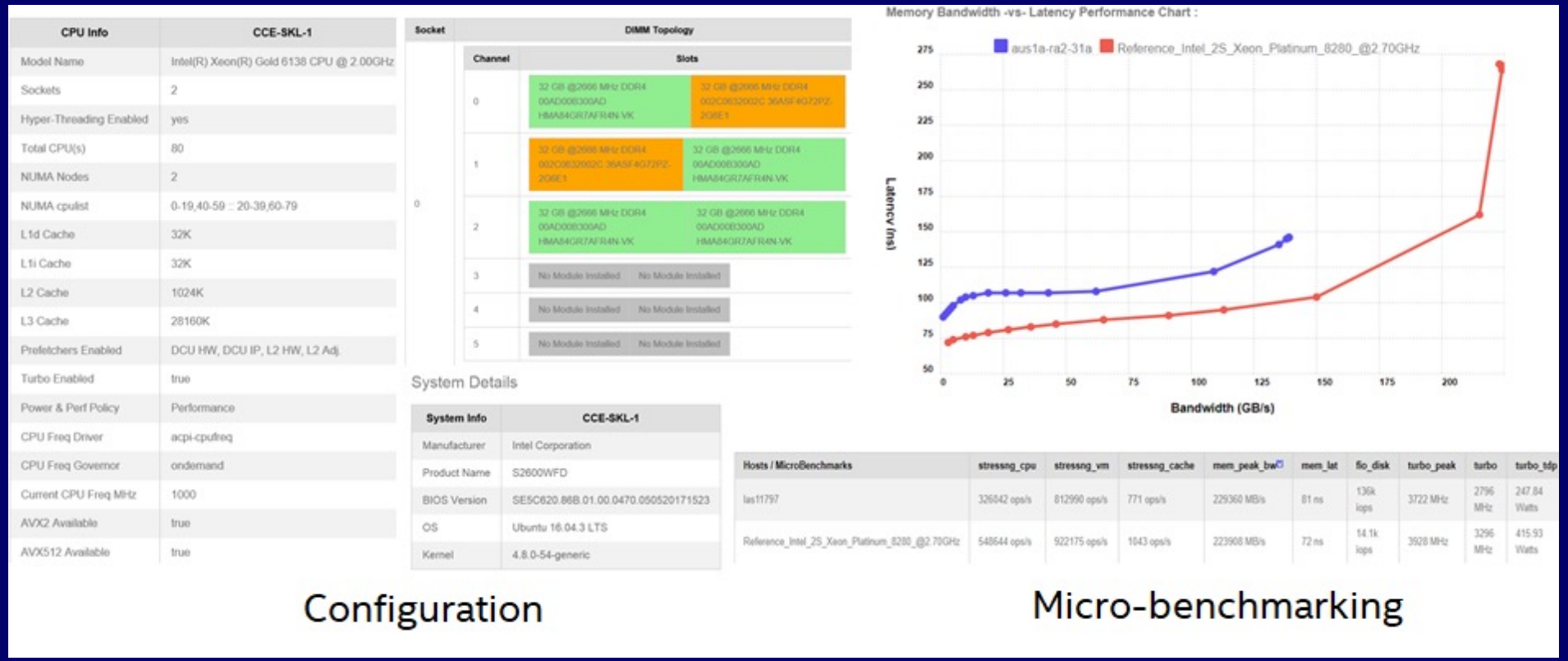
- Evaluate sizing/TCO etc through software emulation of processor SKUs automatically
- Avoid manual intervention, errors, lack of availibility and productivity concerns



Server-Info (svr_info)

Full health report on base system hardware and platform software and knob-settings

- 50% of issues attributed to environment and misconfigurations
- 2 Modes: Configuration, microbenchmarking to test system capabilities



CPU Info	CCE-SKL-1
Model Name	Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz
Sockets	2
Hyper-Threading Enabled	yes
Total CPU(s)	80
NUMA Nodes	2
NUMA cpulist	0-19,40-59 :: 20-39,60-79
L1d Cache	32K
L1i Cache	32K
L2 Cache	1024K
L3 Cache	28160K
Prefetchers Enabled	DCU HW, DCU IP, L2 HW, L2 Adj.
Turbo Enabled	true
Power & Perf Policy	Performance
CPU Freq Driver	acpi-cpufreq
CPU Freq Governor	ondemand
Current CPU Freq MHz	1000
AVX2 Available	true
AVX512 Available	true

Socket	DIMM Topology	
	Channel	Slots
0	0	32 GB @2666 MHz DDR4 00AD00B300AD HMA84GR7AFR4N-VK / 32 GB @2666 MHz DDR4 002C0632002C 36ASF4G72PZ-2G6E1
	1	32 GB @2666 MHz DDR4 002C0632002C 36ASF4G72PZ-2G6E1 / 32 GB @2666 MHz DDR4 00AD00B300AD HMA84GR7AFR4N-VK
	2	32 GB @2666 MHz DDR4 00AD00B300AD HMA84GR7AFR4N-VK / 32 GB @2666 MHz DDR4 00AD00B300AD HMA84GR7AFR4N-VK
	3	No Module Installed / No Module Installed
	4	No Module Installed / No Module Installed
	5	No Module Installed / No Module Installed

System Details

System Info	CCE-SKL-1
Manufacturer	Intel Corporation
Product Name	S2600WFD
BIOS Version	SE5C620.86B.01.00.0470.050520171523
OS	Ubuntu 16.04.3 LTS
Kernel	4.8.0-54-generic

Configuration

Hosts / MicroBenchmarks	stressng_cpu	stressng_vm	stressng_cache	mem_peak_bw	mem_lat	fio_disk	turbo_peak	turbo	turbo_tdp
las1f797	326842 ops/s	812990 ops/s	771 ops/s	229360 MB/s	81 ns	136k iops	3722 MHz	2796 MHz	247.84 Watts
Reference_Intel_2S_Xeon_Platinum_8280_@2.70GHz	548644 ops/s	922175 ops/s	1043 ops/s	223908 MB/s	72 ns	14.1k iops	3928 MHz	3296 MHz	415.93 Watts

Micro-benchmarking

Journey thus far, and continuing . . .

Reduced –

- Elapsed time in half
- Processor utilization by 5X (opened up scaling headroom)
- System time by 50X
- User time by 4X

Continuing analysis: Whether invocation differences (Numpy -> MKL) and (Cpp -> MKL) cause any differences in Cycles/Instruction.

Summary

Critical to divide and conquer but without losing the overall connective tissue between performance critical "reproducers"

Watch out for inter-layer subtleties: crossing languages, frameworks, user-&-kernel, container-&-host

Predictable, small growth in latency is a key requirement for scalable performance.

Intel builds and delivers cloud-ready optimized software, including optimization tools. https://www.intel.com/content/www/us/en/developer/overview.html and further delivers cloud provider specific optimizations as intergrated and tested open source stacks.

Thank you

Q/A