Multiarchitecture Programming for Accelerated Compute, Freedom of Choice for Hardware

Intel® oneAPI Base and HPC Toolkits

Diagnostics & Profiling Tools

Dr. Rafael Lago

intel

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Outline

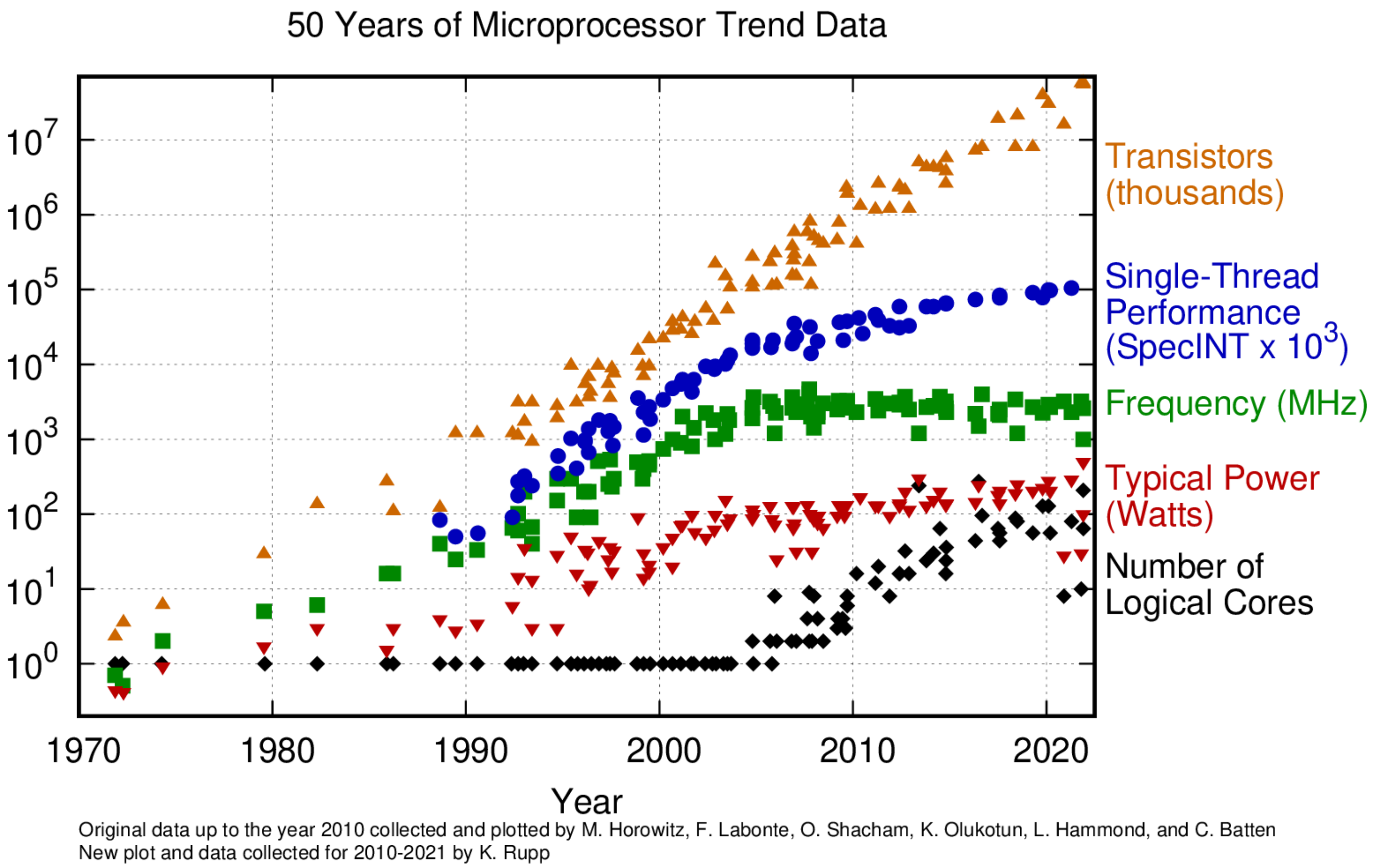
- Overview of Demo Codes
- Intel® Application Performance Snapshot
- Intel® MPI Tuner
- Intel® Trace Analyzer and Collector
- Coffee Break
- Intel® Vtune
- Intel® Advisor

Welcome in the Parallelism Era!

Performance engineering responsibility shifted over the years.

Before:
computer architect

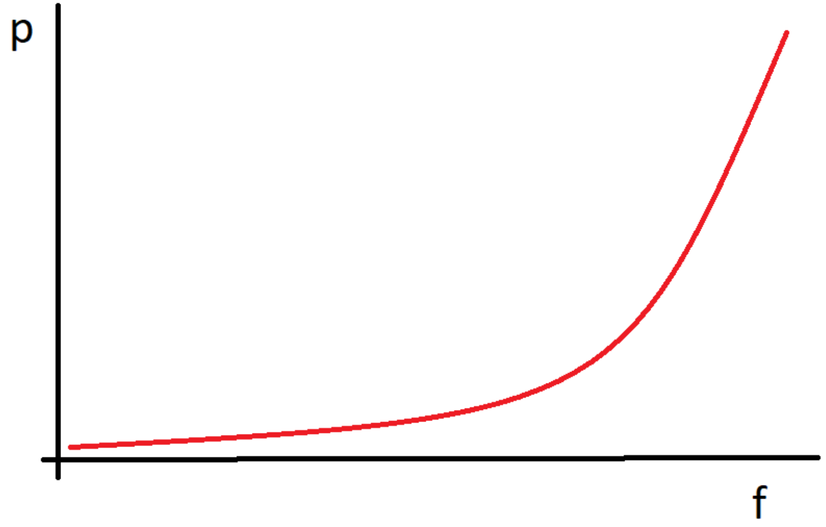
Now:
computer architect
+ software developer



Plot by Karl Rupp: https://github.com/karlrupp/microprocessor-trend-data

Welcome in the Parallelism Area !

Microprocessor frequency over Time (history)



Microprocessor frequency versus Power consumption

... performance is not only the computer architect's job anymore

... performance increase is increasingly the job of the software developer

Opening Statement

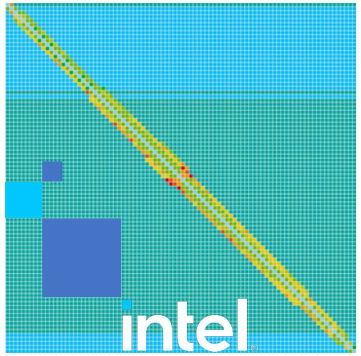
"Parallelism => Performance"

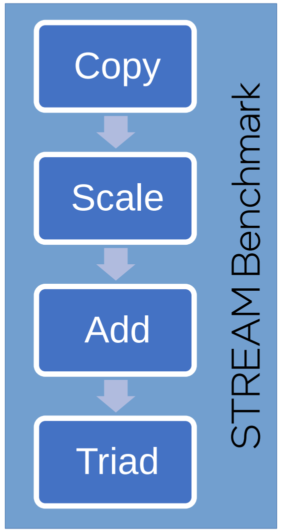
(leads to)

Optimization – *making sure the above statement is true!*

Today's Demos

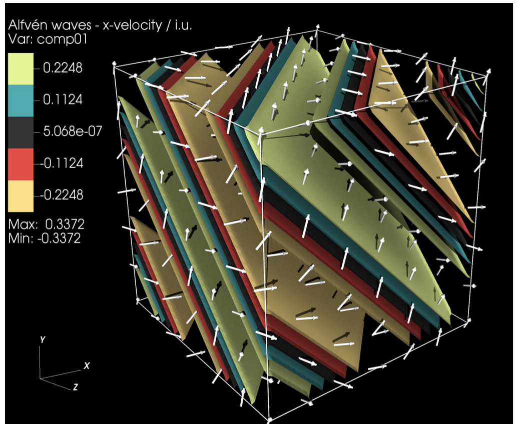
IMB: Intel® MPI Benchmark, a set of microbenchmarks for testing bandwidth & performance of MPI in different configurations

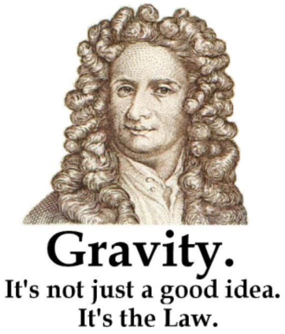




STREAM: four operations benchmark (add, scale, copy and triad) for memory profiling

DPEcho: Data Parallel Eulerian Conservative High Order (DPEcho) for General-Relativity-Magneto-Hydrodynamic simulation (GR-MHD) to model turbulence, wave propagation, stellar winds and processes around black holes





Nbody: Calculates the position of particles using Newton's Law.

- *Further reading:* https://www.intel.com/content/www/us/en/developer/articles/technical/dpecho-general-relativity-sycl-for-2020-beyond.html

Demo 1 – IMB, Intel® MPI (micro)Benchmarks

- Useful for measuring performance/bandwidth for specific MPI settings
- Written in C, sources in $I_MPI_ROOT/benchmarks/imb/
- Some options:
  - Alltoall: name of the test (more here ---->)
  - -npmin A: runs for #ranks=A and larger (powers of 2)
  - -msglog D:E: tests message sizes from 2^D to 2^E
- Example:

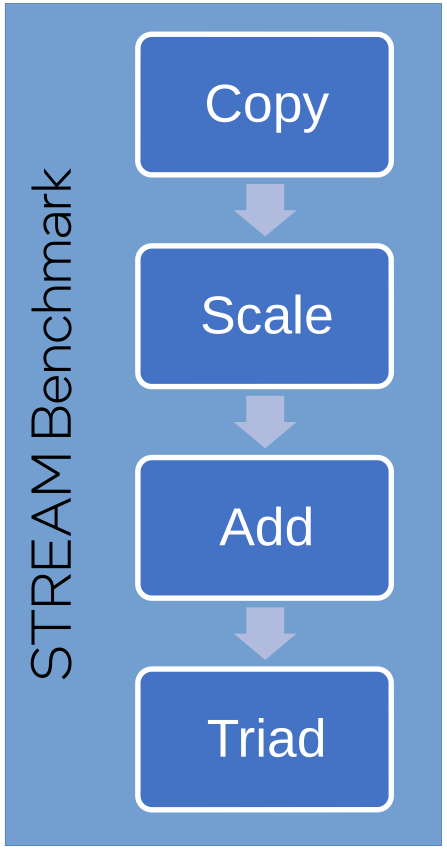
```
mpirun IMB-MPI1 Alltoall -npmin 18 -iter 100 -iter_policy off -msglog 21:21
```

Standard Mode	Multiple Mode
PingPong	Multi-PingPong
PingPongSpecificSource (excluded by default)	Multi-PingPongSpecificSource (excluded by default)
PingPongAnySource (excluded by default)	Multi-PingPongAnySource (excluded by default)
PingPing	Multi-PingPing
PingPingSpecificSource (excluded by default)	Multi-PingPingSpecificSource (excluded by default)
PingPingAnySource (excluded by default)	Multi-PingPingAnySource (excluded by default)
Sendrecv	Multi-Sendrecv
Exchange	Multi-Exchange
Uniband	Multi-Uniband
Biband	Multi-Biband
Bcast	Multi-Bcast
Allgather	Multi-Allgather

Learn more: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-mpi-benchmarks.html

Demo 2 - STREAM Benchmark

John D. McCalpin (TACC)




```
#pragma omp parallel for
   for (j=0; j<STREAM_ARRAY_SIZE; j++)
      c[j] = a[j];
```

0 Flop / 2 * 8 Bytes = 0

```
#pragma omp parallel for
   for (j=0; j<STREAM_ARRAY_SIZE; j++)
      b[j] = scalar*c[j];
```

1 Flop / 3 * 8 Bytes = 0.042

```
#pragma omp parallel for
   for (j=0; j<STREAM_ARRAY_SIZE; j++)
      c[j] = a[j]+b[j];
```

1 Flop / 3 * 8 Bytes = 0.042

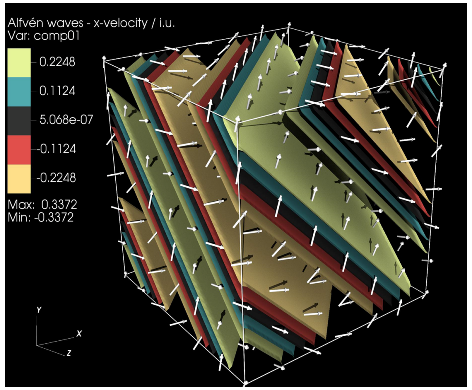
```
#pragma omp parallel for
   for (j=0; j<STREAM_ARRAY_SIZE; j++)
      a[j] = b[j] + scalar*c[j];
```

2 Flop / 4 * 8 Bytes = 0.0625

arithmetic intensity $\gtrsim 9 \Longrightarrow$ compute-bound (dp)
arithmetic intensity $\lesssim 9 \Longrightarrow$ memory-bound

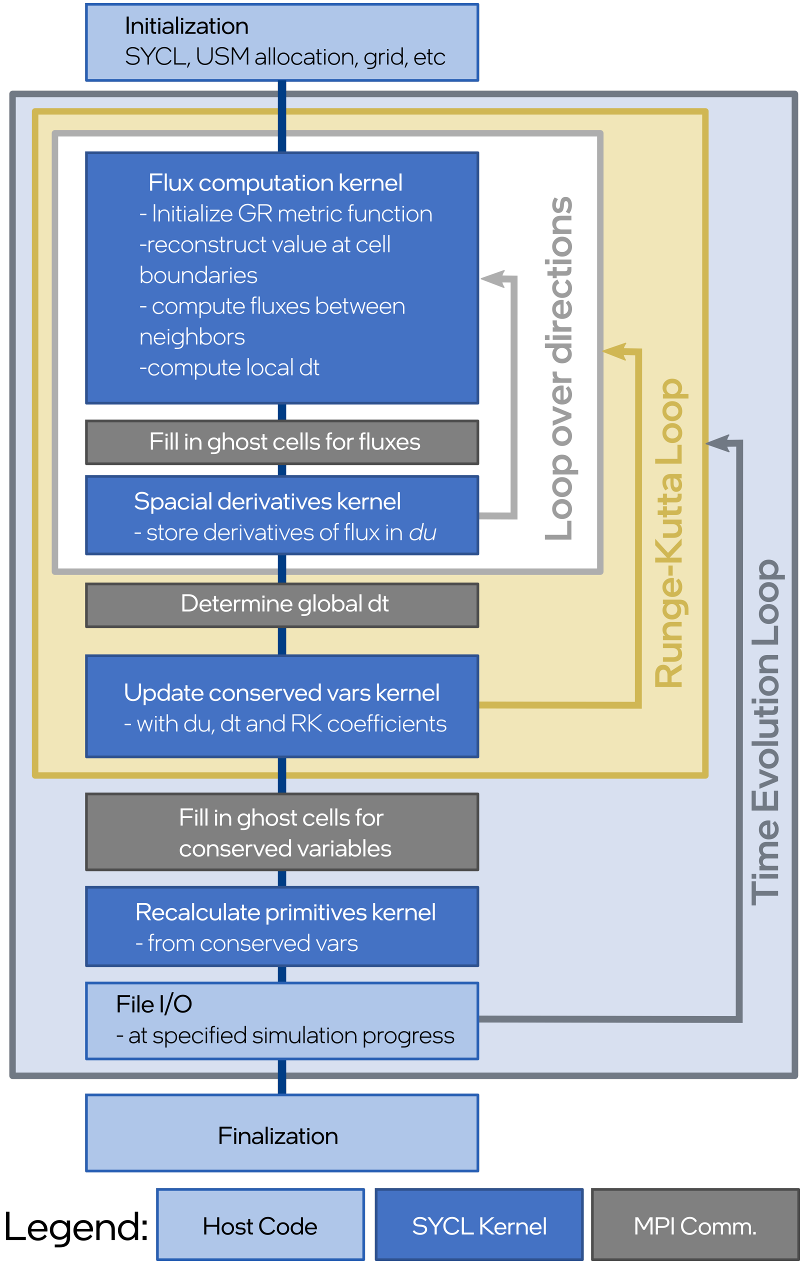
**reports best bandwidth rate out of 10 iterations*

Demo 3 - DPEcho



Data **P**arallel **E**ulerian **C**onservative **H**igh **O**rder (DPEcho) for General-Relativity-Magneto-Hydrodynamic simulation (GR-MHD) to model turbulence, wave propagation, stellar winds and processes around black holes.

- Written in **C++ with MPI+SYCL**
- Three-dimensional cartesian grid discretization
- Adjustable order interpolation of values at boundaries
- 3rd order Runge-Kutta time-stepping scheme
- Hotspots: metric and flux computation, primitive to conserved (and vice-versa) variables conversion, etc



Further reading: https://www.intel.com/content/www/us/en/developer/articles/technical/dpecho-general-relativity-sycl-for-2020-beyond.html

The Seven Levels of Parallelism

- Inter- Node Parallelism
- Intra- Node Parallelism
- Core / Thread Level Parallelism
- Thread-Level Parallelism with SMT
- Instruction-level Parallelism
- Data Level Parallelism
- Accelerator / Offloading Parallelism

Problem Classes

Network

- Message size
- Rank placement
- Load Imbalance
- RTL Overhead
- Network Bandwidth

Cluster/Internode Level

Memory

- False Sharing
- Access with strides
- Latency
- Bandwidth
- NUMA-effect

Threading

- Threaded/serial ratio
- Thread Imbalance
- RTL overhead
- (scheduling, forking)
- Synchronization

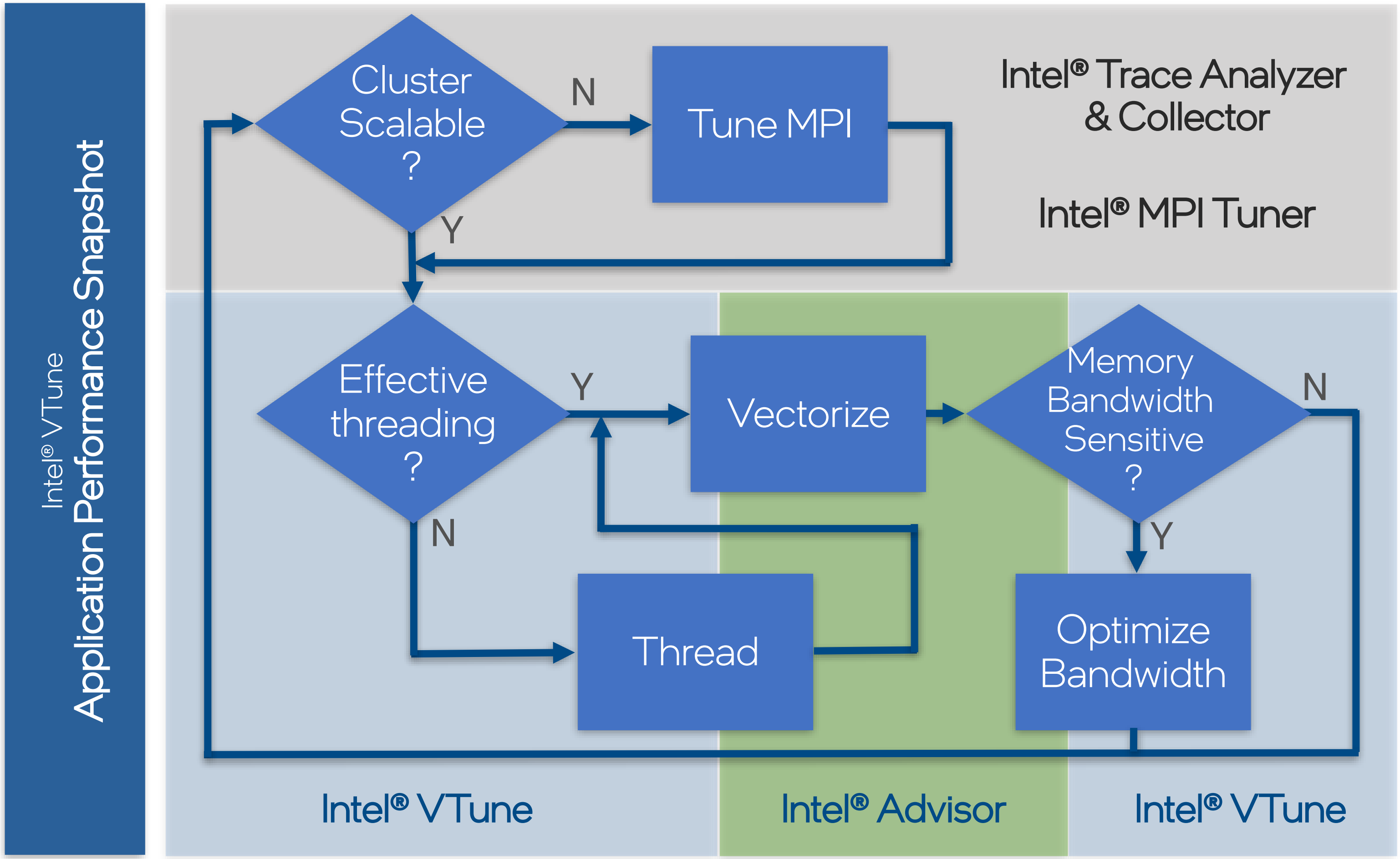
Node/Intranode Level

CPU Core

- µarch issues (IPC)
- Vectorization
- FPU usage efficiency

Core Level

Diagnostics & Profiling Workflow using Intel® oneAPI Base and HPC Toolkits

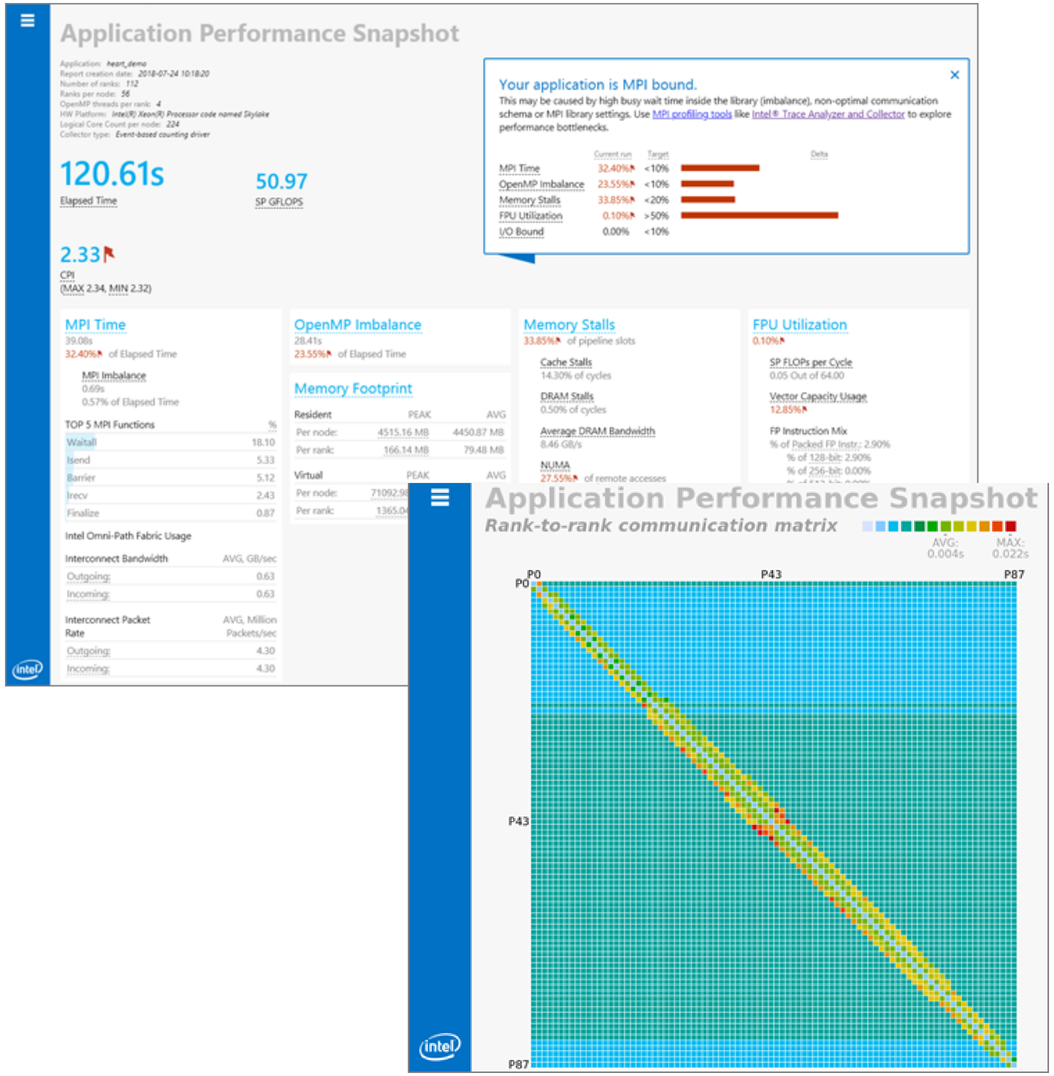


intel.

Intel® Application Performance Snapshot

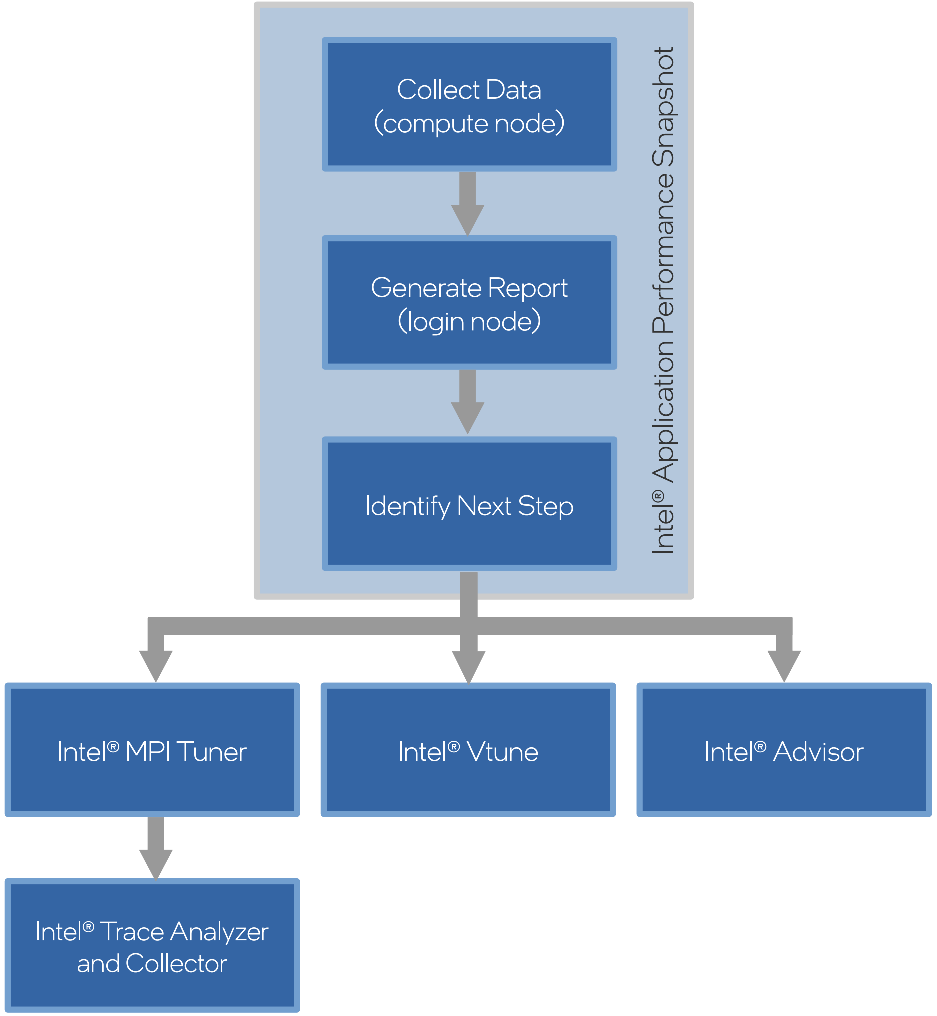
Intel® Application Performance Snapshot

- Part of Intel® VTune
- Lightweight
- *First step to analyze your application!*
- Quick dive into:
  - OpenMP usage
  - MPI balance
  - CPU utilization
  - Memory access efficiency
  - Vectorization
  - I/O
  - Memory footprint
- MPI – friendly
  - Scalable for large workloads
- CLI and HTML reports



Learn more: https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-application-performance-snapshot/top.html

Intel® APS – CLI Essentials



- Data collection:

  source /opt/intel/oneapi/2023.1/setvars.sh

  mpirun <mpi_args> **aps** <aps_args> <my_app+args>

NEW: ***mpirun*** *<mpi_args>* ***-aps*** *<my_app+args>*

- Example of data collection args:

  --collection-mode=<mpi|omp|hwc|all>

  --stat-level=[1-5] (or export APS_STAT_LEVEL)

  --mpi-imbalance=[0-2]

  -r=<results_dir>

- Example report generation:

  aps --report <results_dir> *# summary*

  aps --report **-x** --format=html <results_dir> *# for time r2r matrix*

  aps --report **-x -v** --format=html <results_dir> *# for volume r2r matrix*

Intel® APS – Data Collection Options

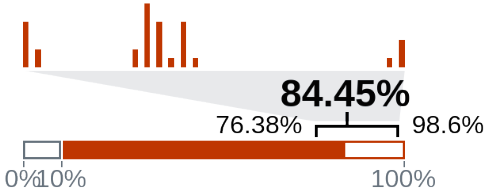
Value	Description
APS_IMBALANCE_TYPE=0	Default value if MPS_STAT_LEVEL=1 Turns off the imbalance calculation.
APS_IMBALANCE_TYPE=1	Default value if MPS_STAT_LEVEL=2 or higher.
APS_IMBALANCE_TYPE=2	Imbalance is calculated by calling MPI_Barrier before any collective operation and measuring the time of the call.

Level	Information is collected about
MPS_STAT_LEVEL=1	MPI functions and their time (defaut).
MPS_STAT_LEVEL=2	MPI functions and amount of transmitted data
MPS_STAT_LEVEL=3	MPI functions, communicators, and message sizes
MPS_STAT_LEVEL=4	MPI functions, communicators, communication directions and aggregated traffic for each direction
MPS_STAT_LEVEL=5	MPI functions, communicators, message sizes, and communication directions

Learn More: https://www.intel.com/content/www/us/en/develop/documentation/application-snapshot-user-guide/top/analyzing-applications/controlling-amount-of-collected-data.html

Intel® APS - Summary Report

MPI Time, % of Elapsed Time



Your application is MPI bound. This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Explore the MPI Imbalance metric if it is available or use MPI profiling tools like Intel® Trace Analyzer and Collector to explore possible performance bottlenecks.

Command:

mpirun aps IMB-MPI1 Allreduce -npmin 32 -iter 100\
-iter_policy off -msglog 21:21
aps --report ./aps_result_*

Application: *IMB-MPI1*
Report creation date: *2023-06-04 07:27:08*
Number of ranks: *32*
Ranks per node: *8*
HW Platform: *Intel(R) Xeon(R) Processor code named Sapphirerapids*
Frequency: *2.00 GHz*
Logical Core Count per node: *224*
Collector type: *Event-based sampling driver,Event-based counting driver*

5.05 s Elapsed Time

2.32 IPC Rate

5.25 SP GFLOPS

0.03 DP GFLOPS

2.88 GHz Average CPU Frequency

Your application might underutilize the available logical CPU cores

because of insufficient parallel work, blocking on synchronization, or too much I/O. Perform function or source line-level profiling with tools like Intel® VTune™ Profiler to discover why the CPU is underutilized.

	Current run	Target	Tuning Potential
MPI Time	84.45%	<10%	
Physical Core Utilization	3.88%	>80%	
Memory Stalls	30.25%	<20%	
Vectorization	98.65%	>70%	

MPI Time

4.02 s
84.45% of Elapsed Time

MPI Imbalance
1 s
21.86% of Elapsed Time

TOP 5 MPI Functions	% of Elapsed Time
MPI_Init_thread	37.54%
MPI_Allreduce	33.76%
MPI_Barrier	8.77%
MPI_Finalize	4.33%
MPI_Comm_split	0.96%

Memory Footprint

Resident
348.06 MB

Resident per Node
2784.5 MB

Virtual
1077196.47 MB

Virtual Per Node
8617571.75 MB

Physical Core Utilization

3.88%

Average Physical Core Utilization
4.33 out of 112 Physical Cores

Memory Stalls

30.25% of Pipeline Slots

Cache Stalls
24.52% of Cycles

DRAM Stalls
5.85% of Cycles

DRAM Bandwidth

Average	9.99 GB/s
Peak	9.8 GB/s
Bound	0%

NUMA
13.62% of Remote Accesses

Vectorization

98.65%

Instruction Mix

SP FLOPs
1.32% of uOps
Packed: 100% from SP FP
128-bit: 100%
256-bit: 0%
512-bit: 0%
Scalar: 0% from SP FP

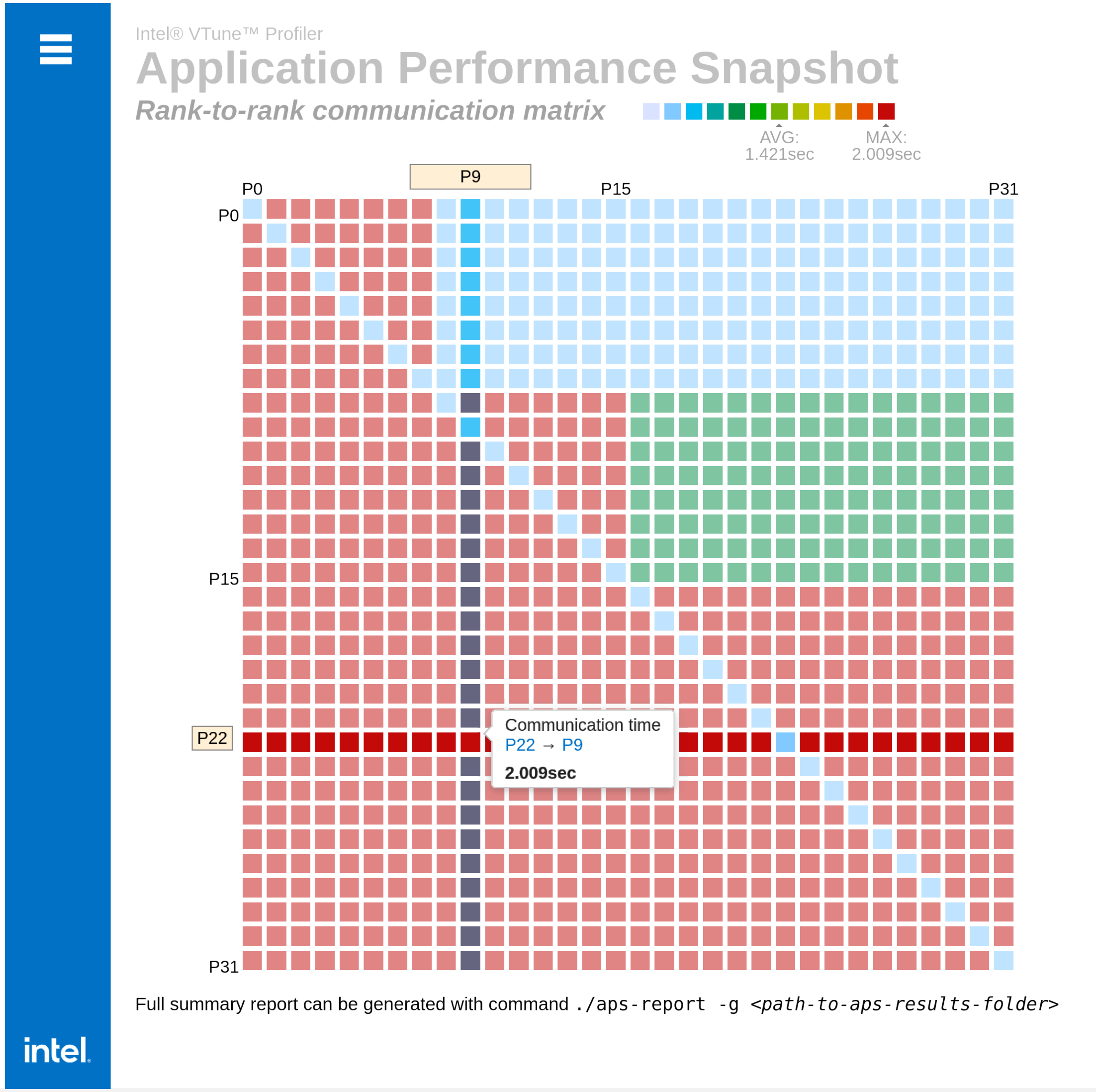
DP FLOPs
0.03% of uOps
Packed: 0% from DP FP
128-bit: 0%
256-bit: 0%
512-bit: 0%
Scalar: 100% from DP FP

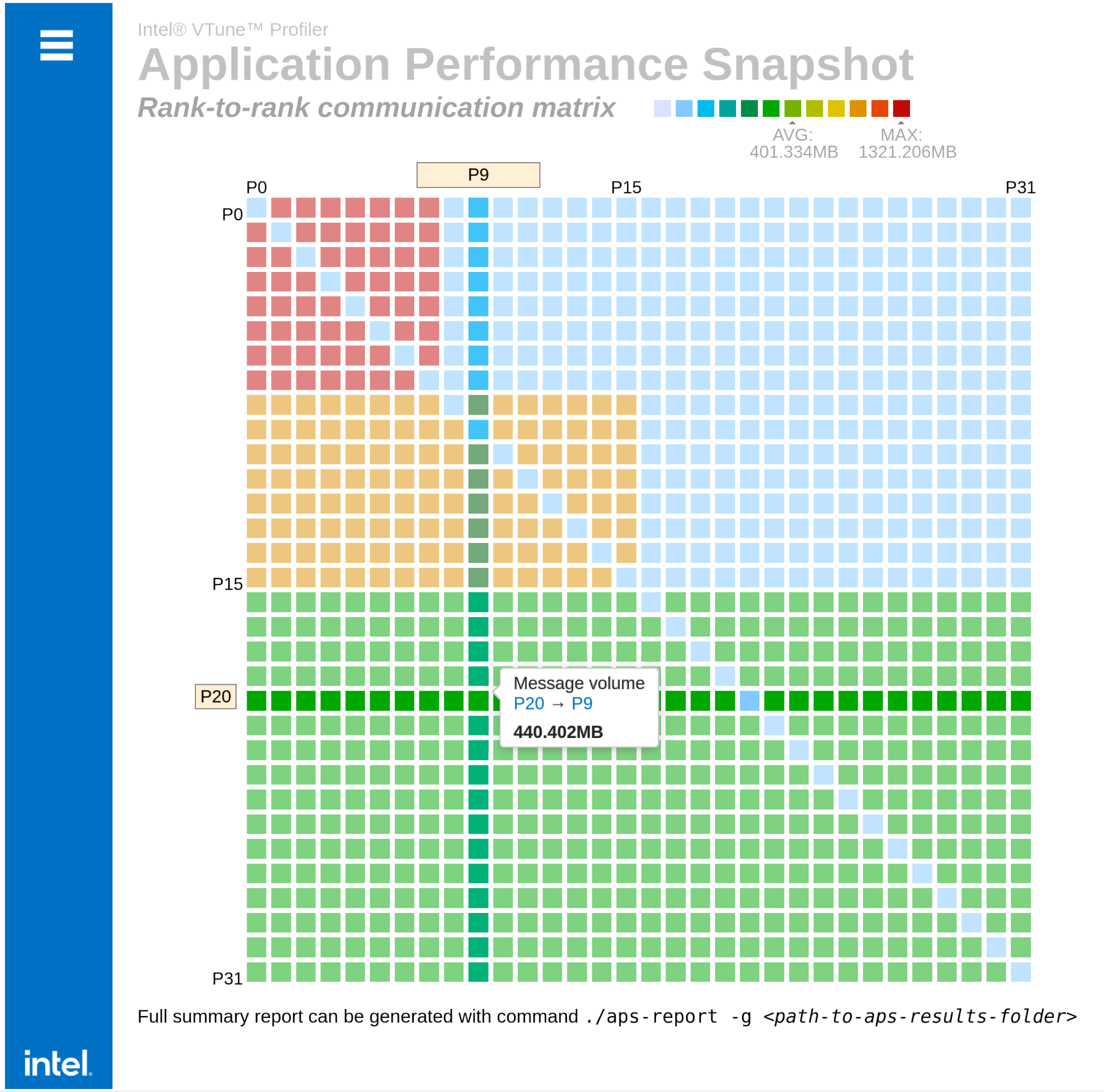
Non-FP
98.62% of uOps

FP Arith/Mem Rd Instr. Ratio
0.04

FP Arith/Mem Wr Instr. Ratio
0.08

Intel® APS - Rank-to-Rank Matrix





Intel® APS – Instrumenting Code Regions

- MPI instrumentation example (Fortran, C, C++):

  call MPI_PControl(5)

  call my_function(args)

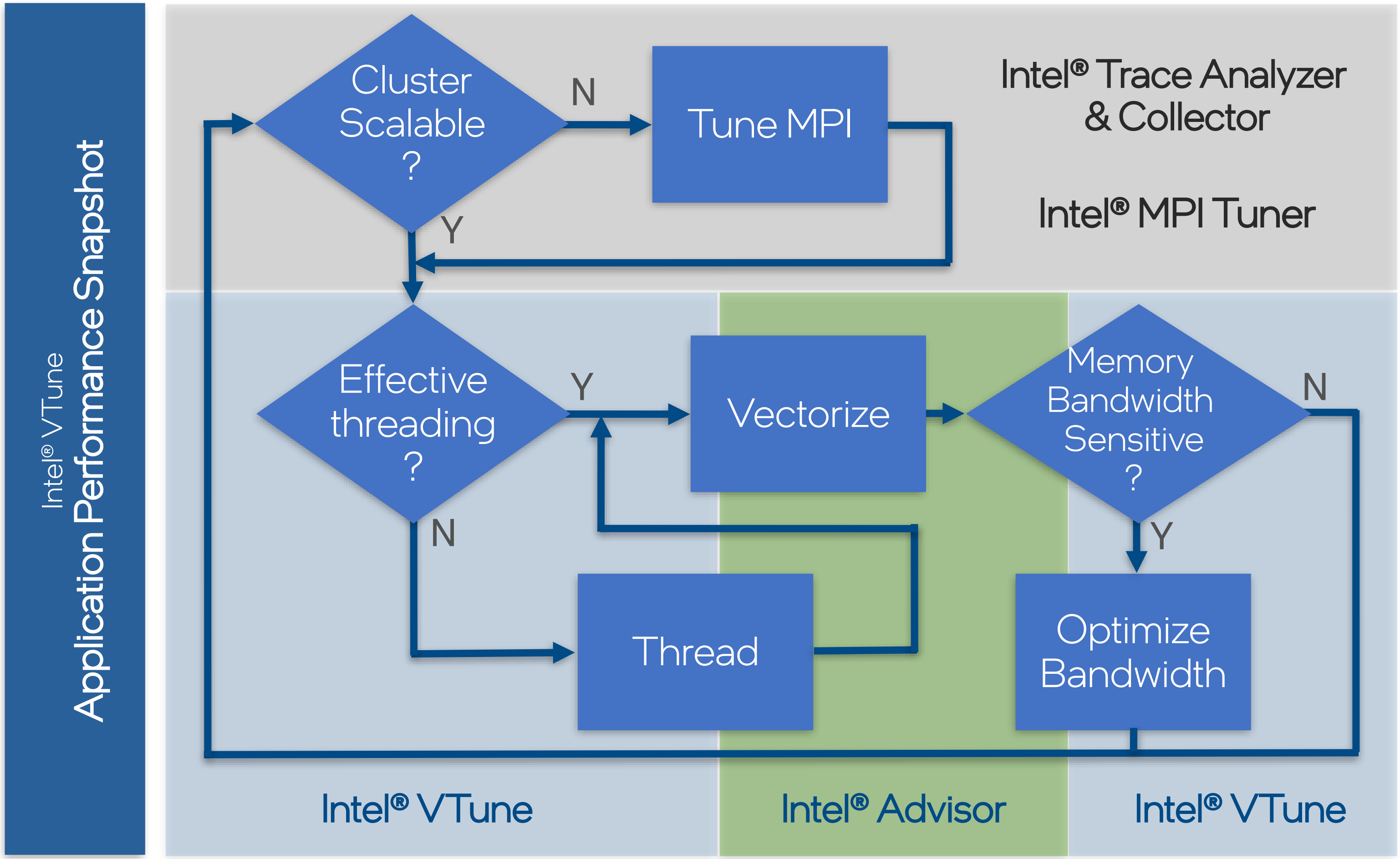
  call MPI_Pcontrol(-5)

  => one APS output folder just for "Region 5"

- MPI_PControl(0) pauses all collection
- MPI_PControl(1) resumes all collection
- regions 2 to 4 are reserved
- For non-MPI, use the ITT API*

*ITT API: https://www.intel.com/content/www/us/en/develop/documentation/application-snapshot-user-guide/top/analyzing-applications/controlling-amount-of-collected-data.html

Intel® APS – Next Step?



Intel® MPI Tuner

Intel® MPI - Pinning Simulator

- Web-based interface
- Platform configuration options
  - load output from **`cpuinfo`** (IMPI utility)
  - or manually define configuration
- Provides IMPI environment variable settings for desired pinning

See more: https://software.intel.com/content/www/us/en/develop/articles/pinning-simulator-for-intel-mpi-library.html

I_MPI_ADJUST_* Family

- Environment variables for selecting the algorithm
  - No recompilation!
- **Performance** depends on
  - Hardware
  - Message Size
  - Number of MPI ranks
  - Topology
- Intel® MPI provides a default setting which should be **performant for most cases**

`I_MPI_ADJUST_ALLREDUCE`	`MPI_Allreduce`	1. Recursive doubling 2. Rabenseifner's 3. Reduce + Bcast 4. Topology aware Reduce + Bcast 5. Binomial gather + scatter 6. Topology aware binominal gather + scatter 7. Shumilin's ring 8. Ring 9. Knomial 10. Topology aware SHM-based flat 11. Topology aware SHM-based Knomial 12. Topology aware SHM-based Knary
`I_MPI_ADJUST_ALLTOALL`	`MPI_Alltoall`	1. Bruck's 2. Isend/Irecv + waitall 3. Pair wise exchange 4. Plum's
`I_MPI_ADJUST_BARRIER`	`MPI_Barrier`	1. Dissemination 2. Recursive doubling 3. Topology aware dissemination 4. Topology aware recursive doubling 5. Binominal gather + scatter 6. Topology aware binominal gather + scatter 7. Topology aware SHM-based flat 8. Topology aware SHM-based Knomial 9. Topology aware SHM-based Knary

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/i-mpi-adjust-family-environment-variables.html

I_MPI_ADJUST_* Family

Custom tuning may be profitable for:

- untested number of ranks configurations
- non-standard message sizes (e.g. 512 KB < msg_size < 1024 KB)
- new network topologies
- untested interconnects
- applications with high imbalance
- non-standard/user defined datatypes
- uncommon collectives (e.g. reduce_scatter)

`I_MPI_ADJUST_ALLREDUCE`	`MPI_Allreduce`	1. Recursive doubling 2. Rabenseifner's 3. Reduce + Bcast 4. Topology aware Reduce + Bcast 5. Binomial gather + scatter 6. Topology aware binominal gather + scatter 7. Shumilin's ring 8. Ring 9. Knomial 10. Topology aware SHM-based flat 11. Topology aware SHM-based Knomial 12. Topology aware SHM-based Knary
`I_MPI_ADJUST_ALLTOALL`	`MPI_Alltoall`	1. Bruck's 2. Isend/Irecv + waitall 3. Pair wise exchange 4. Plum's
`I_MPI_ADJUST_BARRIER`	`MPI_Barrier`	1. Dissemination 2. Recursive doubling 3. Topology aware dissemination 4. Topology aware recursive doubling 5. Binominal gather + scatter 6. Topology aware binominal gather + scatter 7. Topology aware SHM-based flat 8. Topology aware SHM-based Knomial 9. Topology aware SHM-based Knary

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/i-mpi-adjust-family-environment-variables.html

Intel® MPI Tuner – Simple Usage

1) Enable autotuner and store results (store is optional):

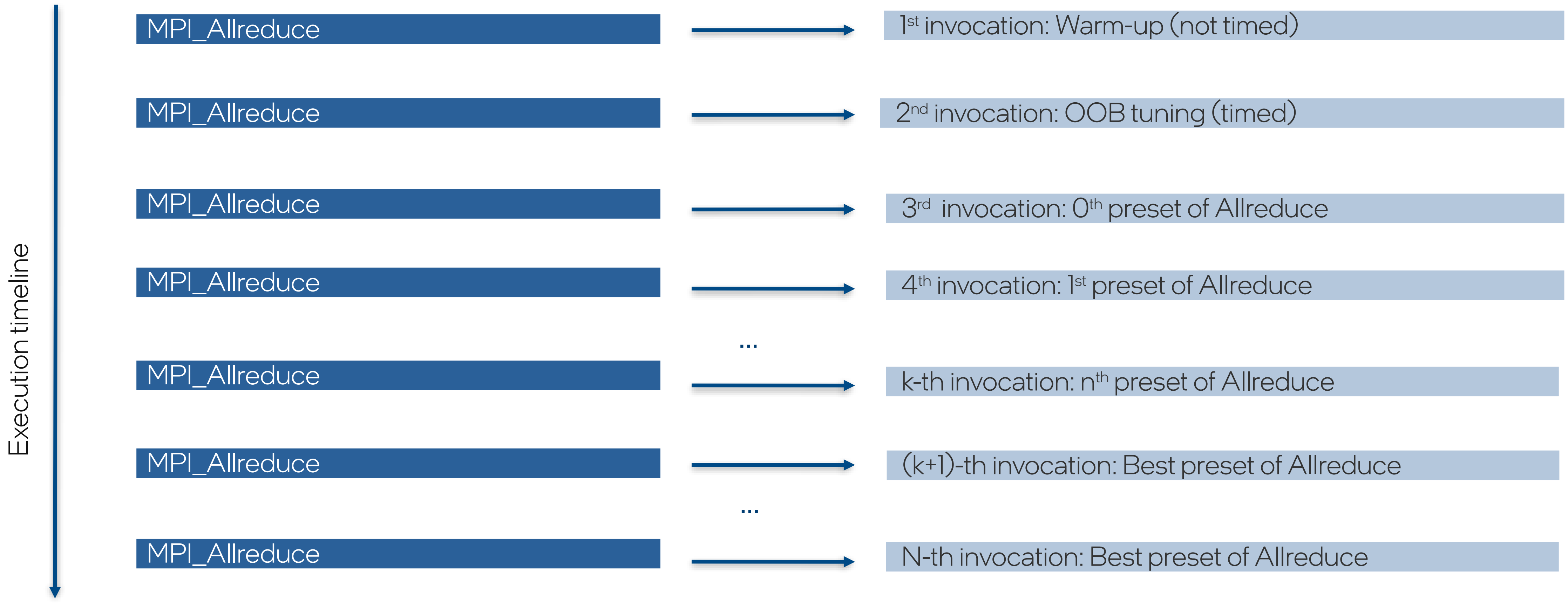
```
export I_MPI_TUNING_MODE=auto
export I_MPI_TUNING_BIN_DUMP=./tuning_results.dat
export I_MPI_TUNNING_AUTO_ITER_NUM=1
mpirun <mpi_args> <app with args>
```

(this run may be slower, due to the tunning)

2) Use the results of autotuner for subsequent launches (optional):

```
unset I_MPI_TUNING_MODE
export I_MPI_TUNING_BIN=./tuning_results.dat
mpirun <mpi_args> <app with args>
```

Intel® MPI Tuner – Simple Usage



(performed for each message size/communicator)

Intel® MPI Tuner – More Options

- **I_MPI_TUNING_MODE=<auto|auto:application|auto:cluster>** (disabled by default)
- **I_MPI_TUNING_AUTO_POLICY=<min|max|avg>** Which metric to use to select best algorithm (max by default)
- **I_MPI_TUNING_AUTO_SYNC=<0|1>** Call internal barrier on every tuning iteration (0 by default)
- **I_MPI_TUNING_AUTO_WARMUP_ITER_NUM=<num>** (1 by default)
- **I_MPI_TUNING_AUTO_ITER_NUM=<num>** (1 by default)

Suggestion: min #iter per collective/message size/communicator
*I_MPI_TUNING_AUTO_WARMUP_ITER_NUM + [(range+1)*I_MPI_TUNING_AUTO_ITER_NUM]*

- **I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD=<max_mem>** Controls message size limit (64Kb default)
- **I_MPI_TUNING_AUTO_STORAGE_SIZE=<max_size>** Communicator storage size (512 Kb default)
- Merging tuning files:

  export **I_MPI_TUNING_BIN=tuned1.dat,tuned2.dat** *# more files allowed*
  export **I_MPI_TUNING_BIN_DUMP=tuning_merged.dat**
  mpiexec –n 1 ./dummy_mpi_app

Troubleshooting MPI Applications

Using System's GDB

- Interactive debugging using system's gdb:

$ mpirun -n 4 **-gdb** IMB-MPI1 allreduce

or

$ mpirun -n 4 **–gdba** *<MPI_PID>*

- Starts one gdb-server and one gdb-client per rank. User interacts with gdb-server only.

```
$ mpirun -n 4 -gdb ./mpi_hello_world
mpigdb: attaching to 14395 ./mpi_hello_world reginn
mpigdb: attaching to 14396 ./mpi_hello_world reginn
mpigdb: attaching to 14397 ./mpi_hello_world reginn
mpigdb: attaching to 14398 ./mpi_hello_world reginn
[0-3] (mpigdb) b ./mpi_hello_world.c:37
[0-3]   Breakpoint 1 at 0x401221: file ./mpi_hello_world
[0-3] (mpigdb) r
[0-3]   Continuing.
[1-3]
[0]
[1]     Breakpoint 1, printHello (rank=1, size=4) at ./m
[2]     Breakpoint 1, printHello (rank=2, size=4) at ./m
[3]     Breakpoint 1, printHello (rank=3, size=4) at ./m
[0]     Breakpoint 1, printHello (rank=0, size=4) at ./m
[1-3]   37          MPI_Get_processor_name(name, &namele
[0]     37          MPI_Get_processor_name(name, &namele
[0-3] (mpigdb) s
[3]     PMPI_Get_processor_name (name=0x7fffdc1ad250 "",
[0]     PMPI_Get_processor_name (name=0x7ffe79890710 "",
[1]     PMPI_Get_processor_name (name=0x7ffe916f4680 "",
[2]     PMPI_Get_processor_name (name=0x7ffc3f6ab0d0 "",
[0-3] (mpigdb) s
[0-3]   73      in ../../src/mpi/misc/getpname.c
[0-3] (mpigdb) r
[0-3]   Continuing.
```

Troubleshooting MPI

SLURM's multi-prog

- **srun –multi-prog ./multiprog.conf**
- multiprog.conf example:

```
# <rank-range> <app-with-args>
0-1 ./my_app
2  gdb -- ./my_app
3  vtune -c hpc-performance -- ./my_app
4  vtune -c memory-access -- ./my_app
5  ./my_app
```

Troubleshooting MPI

Checking correctness with ITAC

(Attention: the output can be quite verbose!)

Intel(R) Trace Analyser and Collector Correctness Check:

mpirun -n 4 **-check_mpi** <app>

or

export LD_PRELOAD=${VT_SLIB_DIR}/**libVTmc.so**:\
${I_MPI_ROOT}/lib/release/libmpi.so
srun <app>

```
$ mpirun -n 4 -check_mpi ./mpi_hello_world
(...)
[0] INFO: CHECK GLOBAL:COLLECTIVE:COMM_FREE_MISMATCH ON
[0] INFO: maximum number of errors before aborting: CHEC
[0] INFO: maximum number of reports before aborting: CHE
[0] INFO: maximum number of times each error is reported
[0] INFO: timeout for deadlock detection: DEADLOCK-TIMEO
[0] INFO: timeout for deadlock warning: DEADLOCK-WARNING
[0] INFO: maximum number of reported pending messages: C

Hello world: rank 0 of 4 running on rlago-mobl3

[1] ERROR: LOCAL:MPI:CALL_FAILED: error
[1] ERROR:    Invalid rank has value 100 but must be non
[1] ERROR:    Error occurred at:
[1] ERROR:       MPI_Send(*buf=0x7ffd144f439c, count=1,
[1] ERROR:       printHello (/home/rlago/area51/demo/mpi
[1] ERROR:       main (/home/rlago/area51/demo/mpi/./mpi
[1] ERROR:       (/usr/lib/x86_64-linux-gnu/libc.so.6)
[1] ERROR:       (/usr/lib/x86_64-linux-gnu/libc.so.6)
[1] ERROR:       _start (/home/rlago/area51/demo/mpi/mpi
[1] INFO: 1 error, limit CHECK-MAX-ERRORS reached => abo

[2] ERROR: LOCAL:MPI:CALL_FAILED: error
[2] ERROR:    Invalid rank has value 100 but must be non
[2] ERROR:    Error occurred at:
[2] ERROR:       MPI_Send(*buf=0x7ffc07714cec, count=1,
[2] ERROR:       printHello (/home/rlago/area51/demo/mpi
[2] ERROR:       main (/home/rlago/area51/demo/mpi/./mpi
[2] ERROR:       (/usr/lib/x86_64-linux-gnu/libc.so.6)
[2] ERROR:       (/usr/lib/x86_64-linux-gnu/libc.so.6)
[2] ERROR:       _start (/home/rlago/area51/demo/mpi/mpi
```

Intel® Trace Analyzer and Collector

Intel® Trace Analyzer and Collector

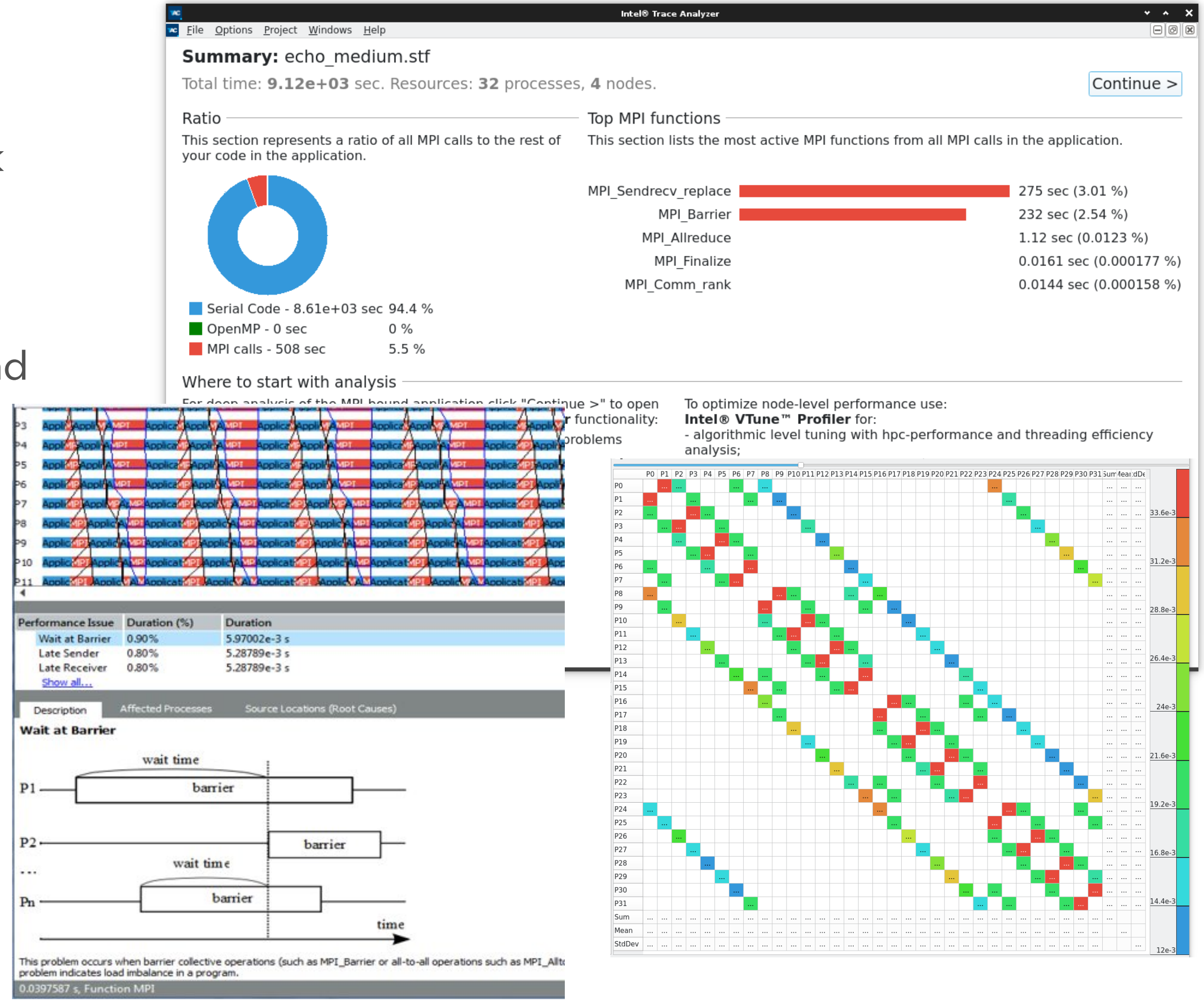
MPI Profiling for Cluster Applications

Understand MPI Application across its full runtime

- Find temporal dependencies and bottlenecksCheck correctness
- Evaluate profiling statistics and load balancing
- learn about communication patterns, parameters and performance data
- Identify communication hot spots
- Instrumentation & Tracing

MPI Checking

- Detect deadlocks, data corruption, error with MPI parameters, data types, buffers, communications, P2P and collective operations
- Scale to extremely large systems



ITAC - Essentials

- Set trace filename: export **VT_LOGFILE_NAME**=<filename.stf>
- Set trace type: export **VT_LOGFILE_FORMAT**=SINGLESTF
- Running with mpirun:

  mpirun **-trace** <app>

- Running with srun:

  export LD_PRELOAD=${VT_SLIB_DIR}/**libVT.so**:${I_MPI_ROOT}/lib/release/libmpi.so
  srun <app>

- By default, only MPI is instrumented. Compile with -tcollect to get trace of full application:

  mpiicx **–tcollect** <app> *# only with legacy compilers :(*

Learn More: *https://www.intel.com/content/www/us/en/develop/documentation/itac-user-and-reference-guide/top/intel-trace-collector-reference/api-reference.html*

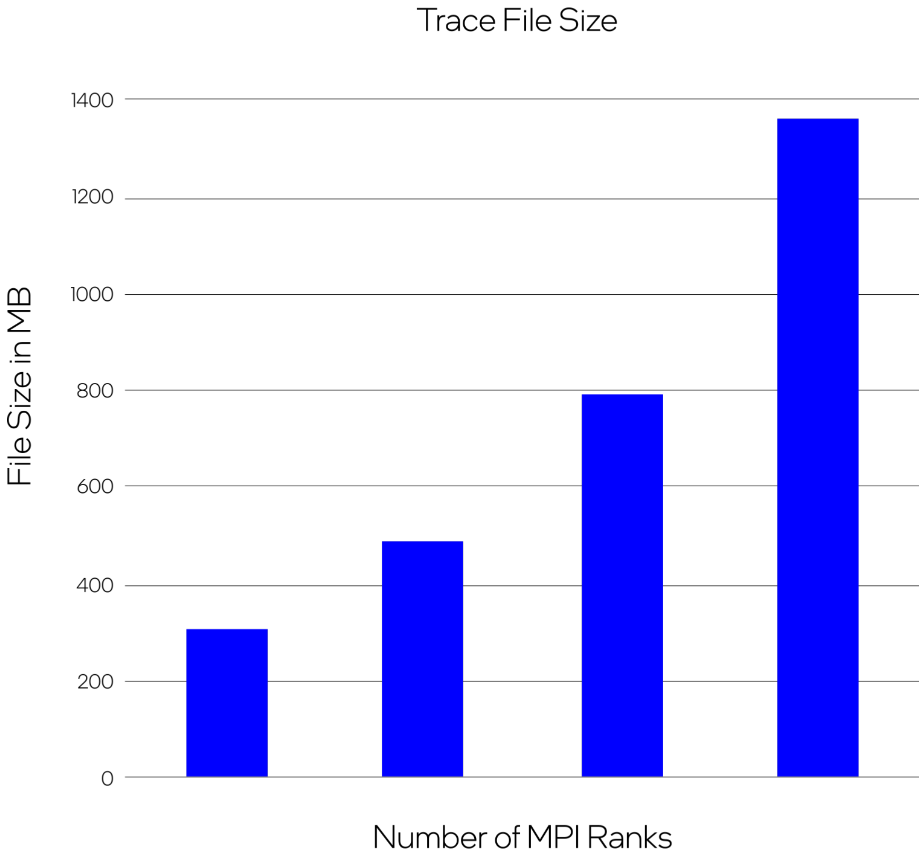
ITAC – Reducing Trace Size

Output size may be MASSIVE! Filter output with:

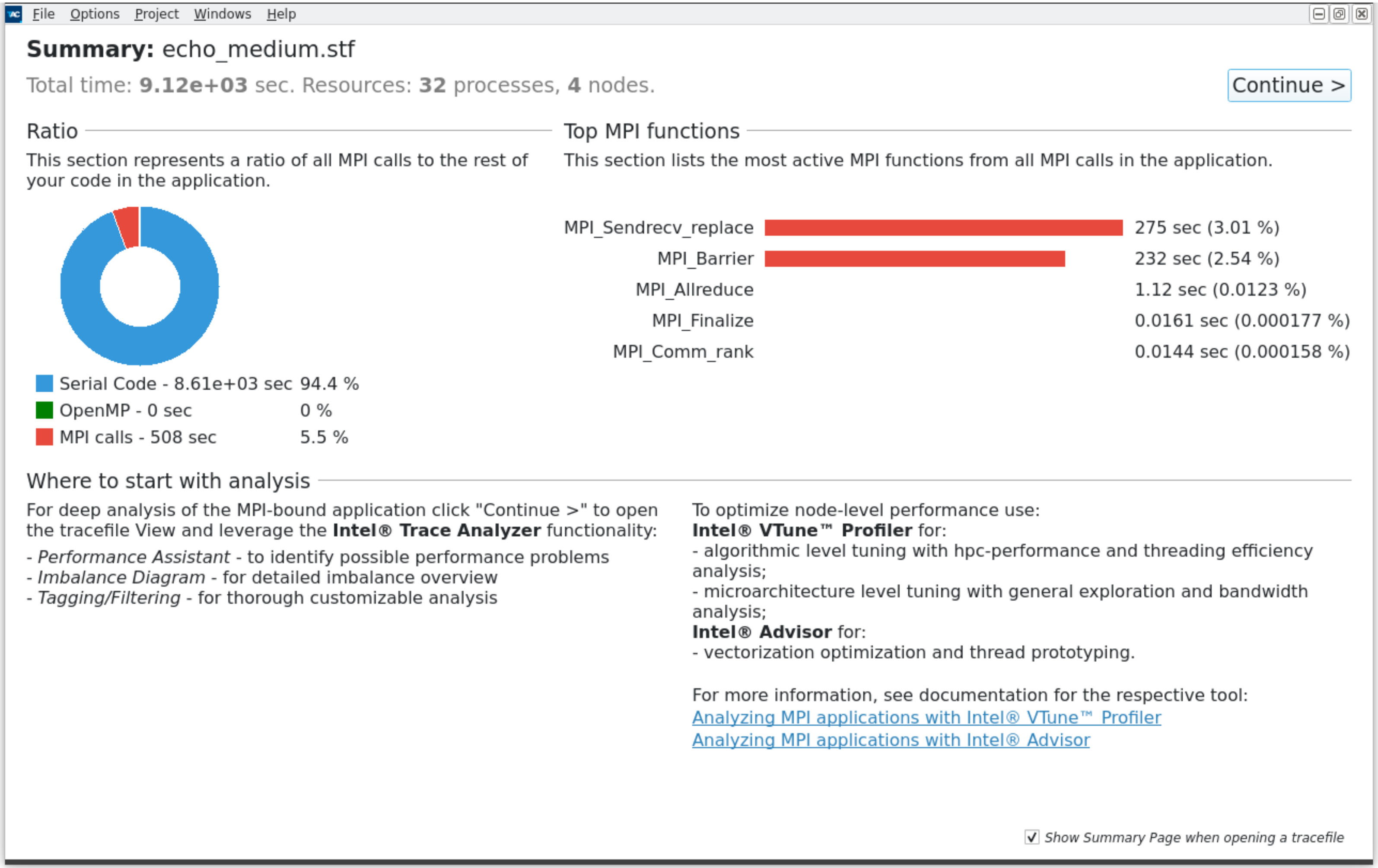
mpirun **-trace-pt2pt** or **-trace-collectives**

Manual code instrumentation via ITAC's API:

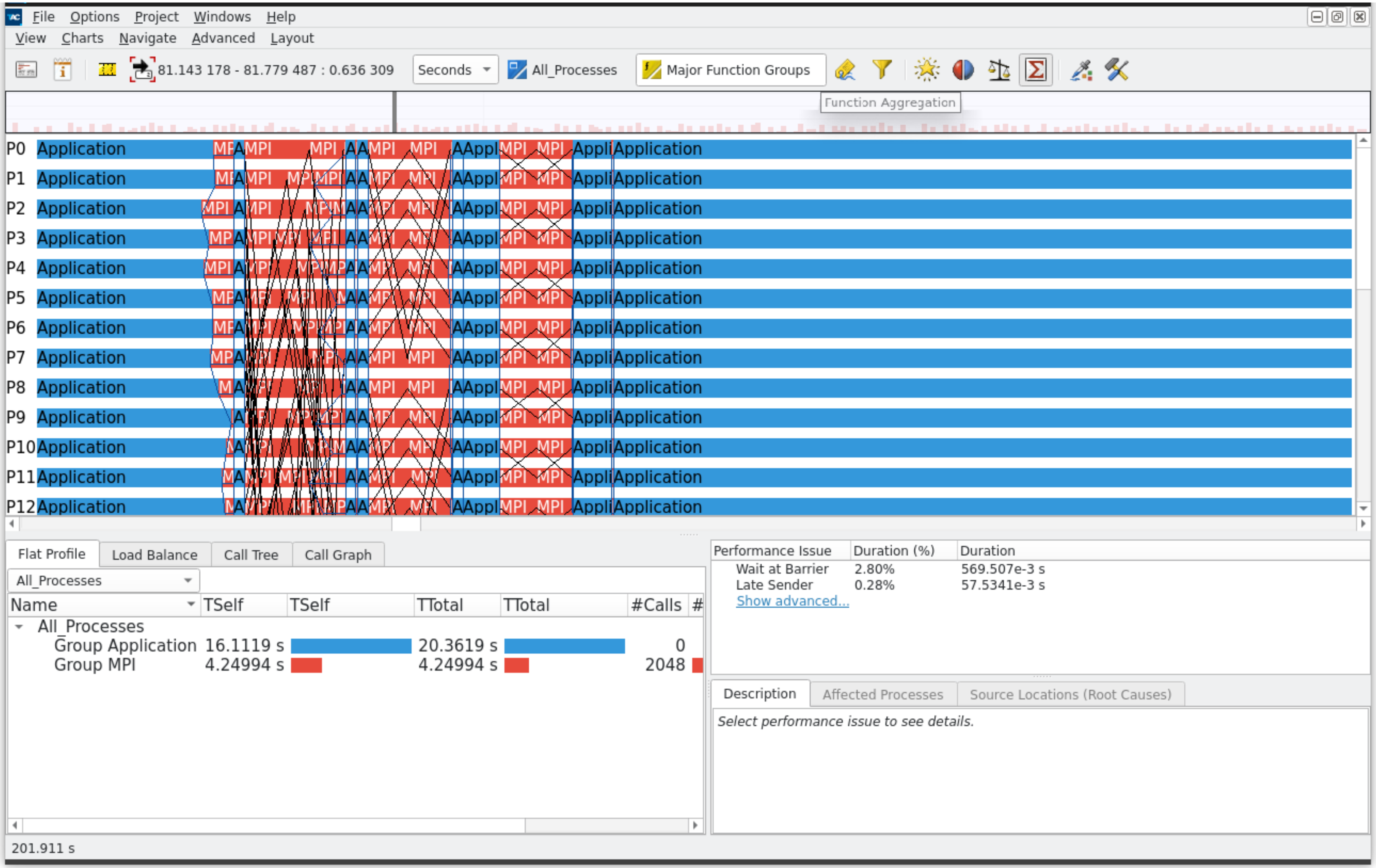
- **VT_initialize/VT_finalize:** initializes the collector
- **VT_traceon/VT_traceoff:** enables/disables trace collection (statistical data is not affected)
- **VT_funcdef, VT_begin, VT_end:** defines a region name, as well as its beginning and end
- Fortran calls: **VTINIT, VTFINII, VTTRACEON, VTTRACEOFF**, etc



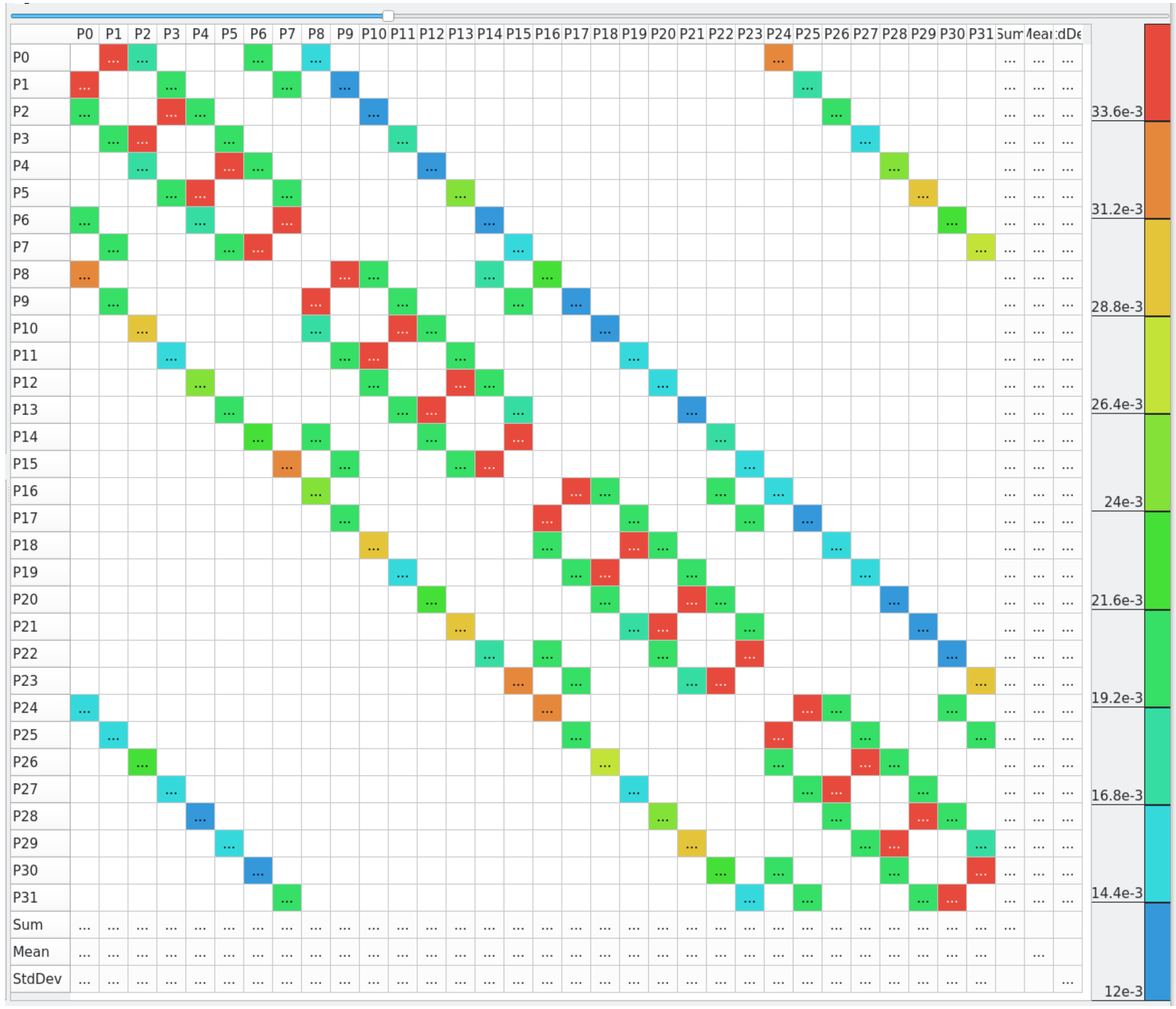
ITAC – DPEcho Summary



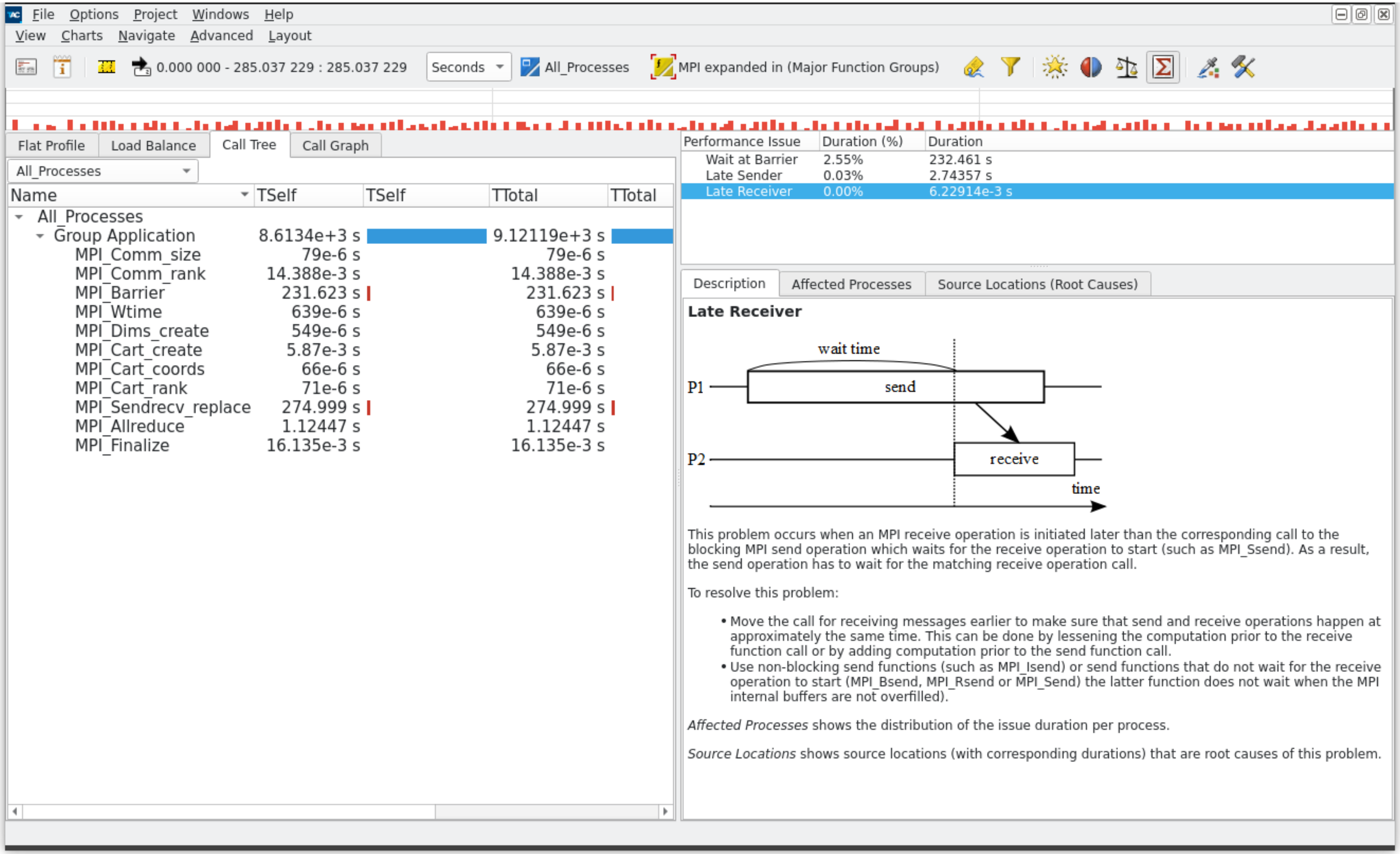
ITAC – DPEcho Event Timeline



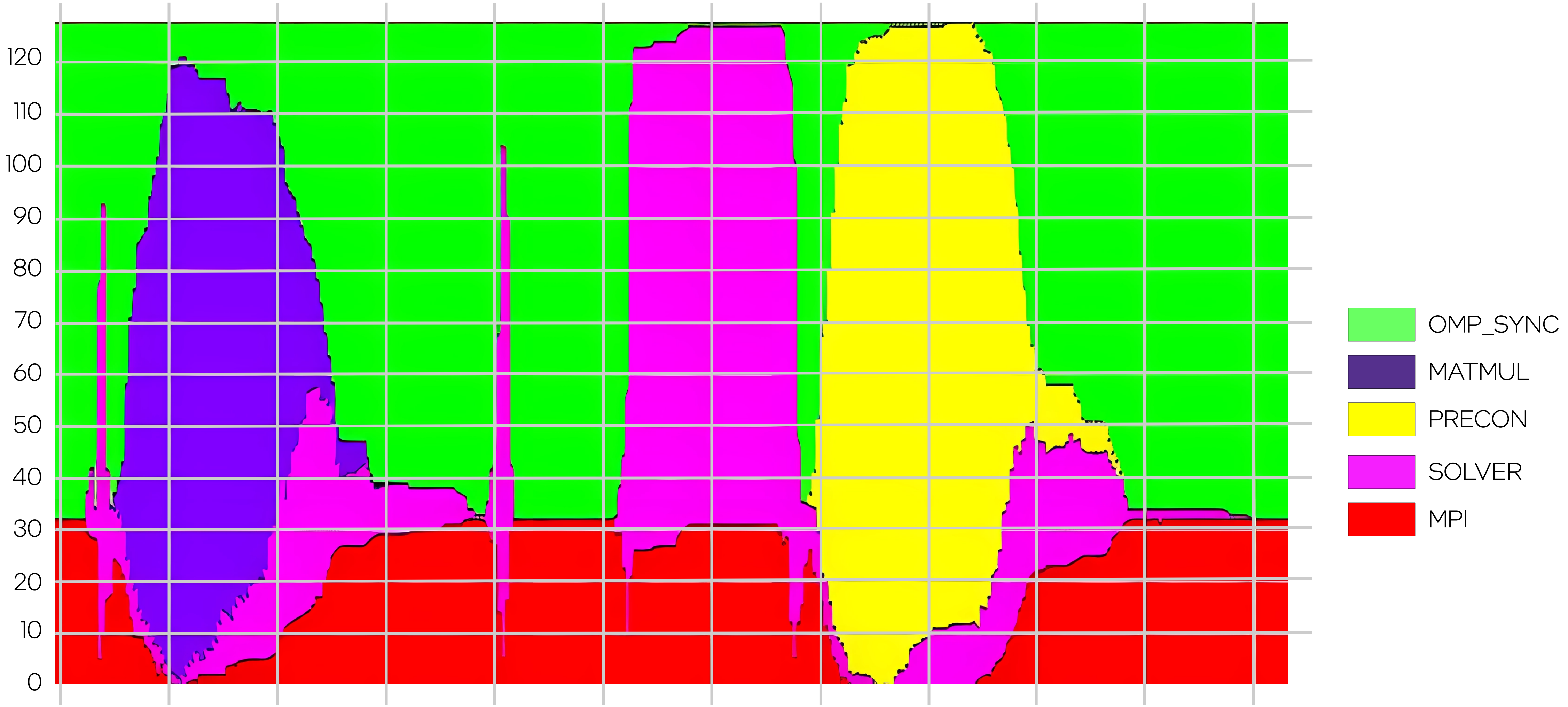
ITAC – DPEcho Message Profile



ITAC – DPEcho Performance Assistant



ITAC – Quantitative Timeline (Sample)

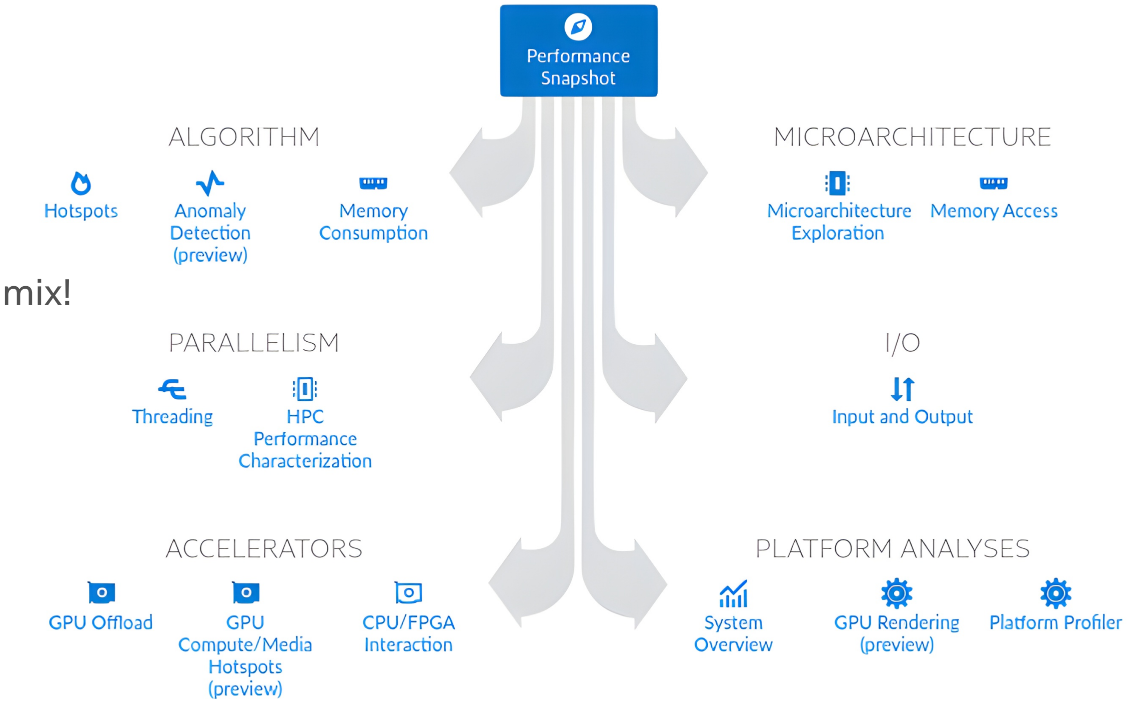


Intel® VTune™ Profiler

Intel® VTune™ Profiler

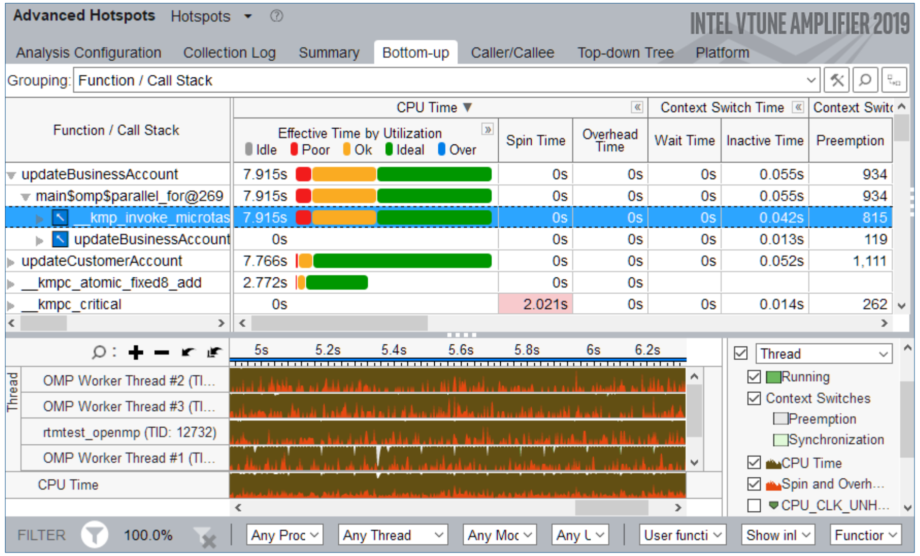
Save time optimizing code

- Accurately profile C, C++, Fortran*, Python*, Go*, Java* or any mix!
- Threading, memory, cache, storage & more
- Save time: rich analysis leads to insight
- Take advantage of [Priority Support]
  - Connects customers to Intel engineers for confidential inquiries (paid versions)



What's new in 2022 Release (selected)

- **Improved Accelerator Profiling**
  - Identify occupancy issues on GPU
  - Identify inefficient code paths between host and device
  - Multiple GPU systems and MPI applications.
- **New Profiles**
  - Flame Graph
  - CPU Throttling analysis
- **Better Data**
  - I/O Analysis with support for MPI applications.
- **New HW Support**
  - Support added for the 3rd Gen Xeon® (i.e. Ice Lake server), Intel® microarchitectures code named Alder Lake and Alchemist (i.e. DG2)



Learn more: https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.htm

intel.

Intel® VTune™ Profiler - Cookbook

- Analyze Common Performance Bottlenecks – C++ Sample Code
- Analyzing an OpenMP+ and MPI Application – C++ Sample Code
- Performance Analysis Cookbook
  - Frequent DRAM Accesses
  - Remote Socket Accesses
  - OpenMP* Imbalance and Scheduling Overhead
  - Profiling in a Docker* Container
  - (…)
- Matrix Multiply:
  - https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/VTuneProfiler/matrix_multiply_vtune
- NBody:
  - https://github.com/oneapi-src/oneAPI-samples/tree/master/DirectProgramming/DPC++/N-BodyMethods/Nbody/

Link: https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/introduction/tutorials-and-samples.html

Two Great Ways to Collect Data

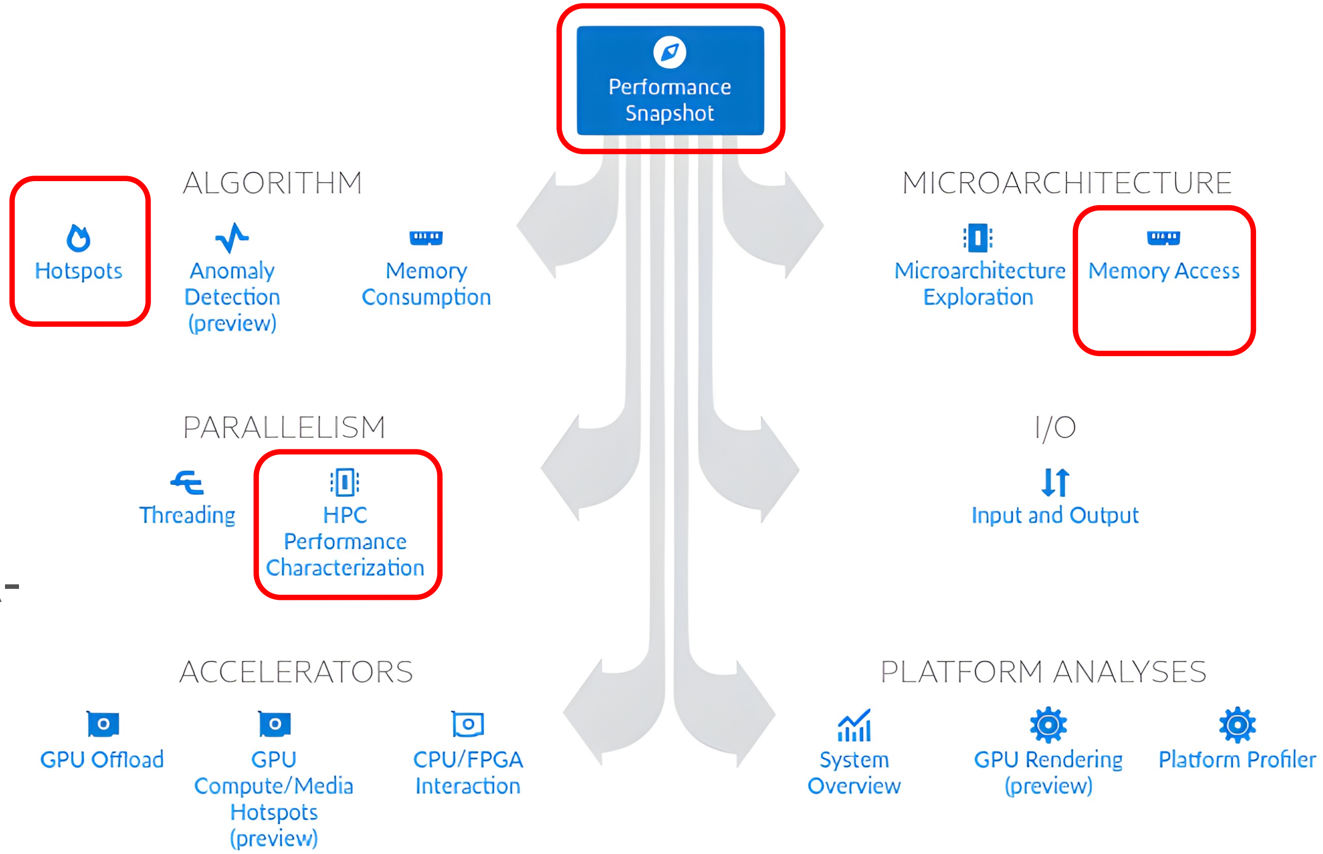
Intel® VTune

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

No recompilation - C, C++, C#, Fortran, Java, Python, Assembly

VTune – Analysis GUI

- **Performance Snapshot:**
  summarizes issues and recommends the next analysis to perform
- **Hotspots:**
  identify time-consuming functions/regions
- **Memory Access*:**
  identifies memory access- and NUMA-related issues
- **HPC*:**
  analyses compute-intensive applications, CPU/GPU utilization, memory efficiency, vectorization, threading, etc*



** requires Intel Sample Drivers or perf_event_paranoid < 2*

VTune – Data Collection Essentials

- **Performance Snapshot:**
  type: -c performance-snapshot

- **Hotspots:**
  type: -c hotspots
  extra: -knob sampling-mode=hw
- **Memory Access*:**
  type: -c memory_access
  extra: -knob analyze-mem-objects=true

- **HPC*:**
  type: -c hpc-performance
  extra: -analyze-system

Command:

```
$ vtune -c <analisys_type> \
        -r <result_path> \
        -- <app>
```

** requires Intel Sample Drivers or perf_event_paranoid < 2*

intel.

VTune – Seamless Remote Analysis

*Not *that* seamless, due to Firewalls, VPNs and etc*

- Open webserver in raven (leave terminal open):
  $ vtune-server --web-port=12347 --allow-remote-access --data-directory=.

- Copy URL offered by Vtune:
  "Serving GUI at https://<some_IP>:12347/?one-time-token=1234af1bc23(…)"

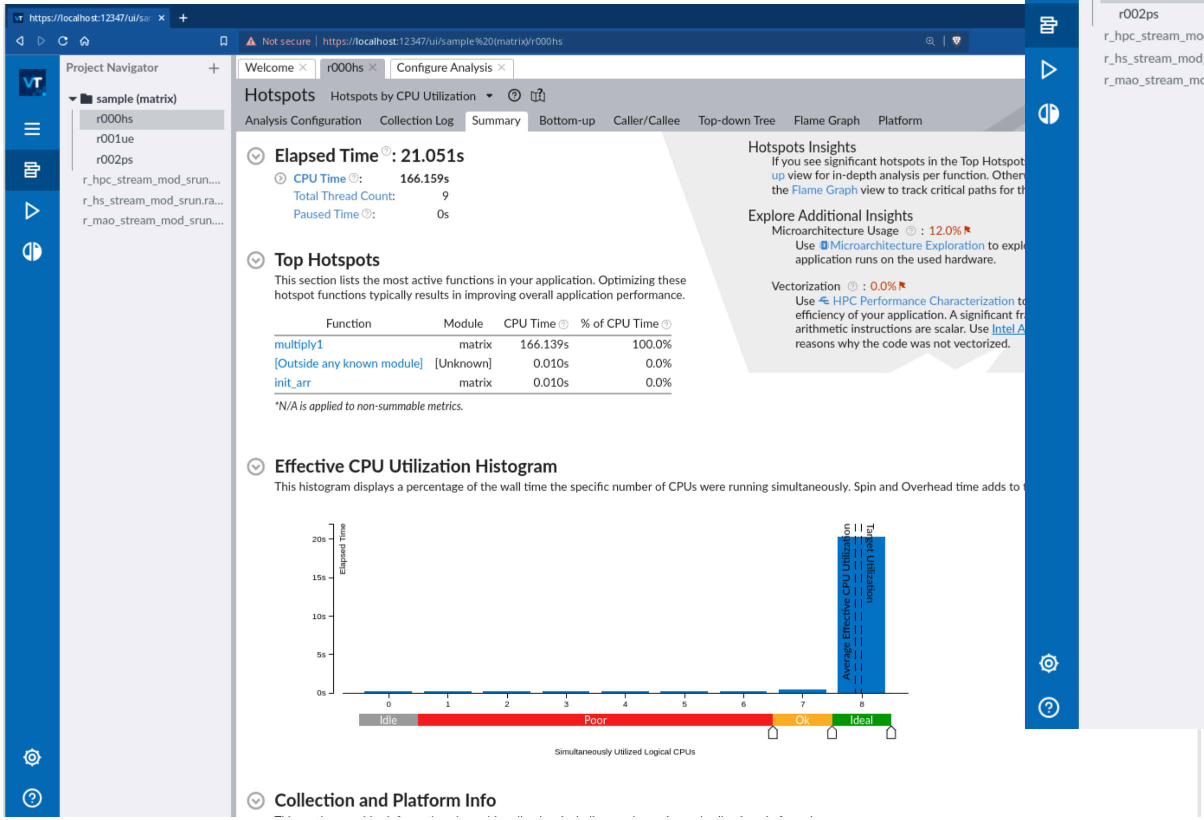
- Create a tunnel from your local machine (leave terminal open):
  $ ssh -L 12347:<some_IP>:12347 <server_address>

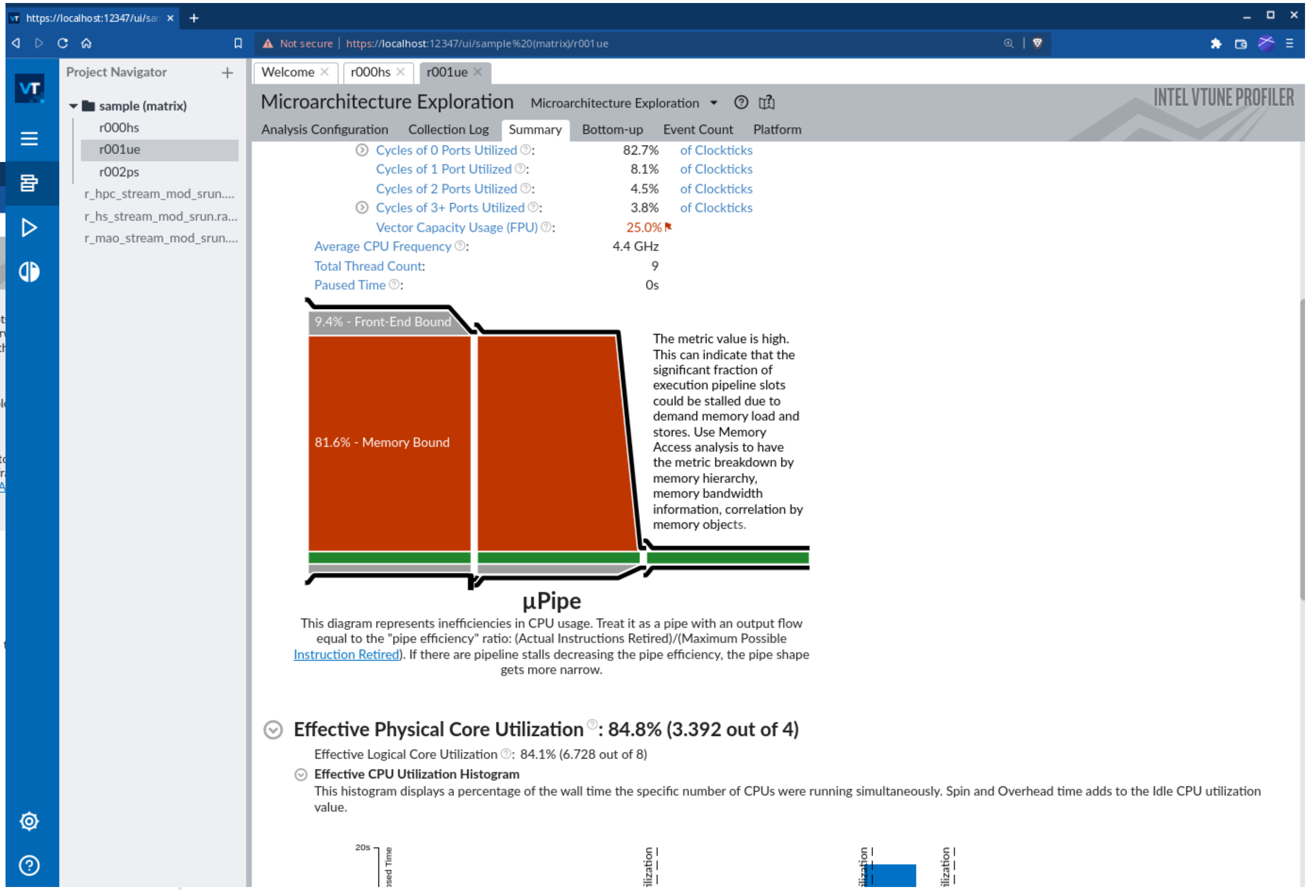
- Open new URL in your local browser, replacing <some_IP> by "localhost"

- Accept the "not secure" connection warning

VTune – Seamless Remote Analysis

*Not *that* seamless, due to Firewalls, VPNs and etc*





STREAM – Baseline vs "Broken"

```
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 1000000000 (elements), Offset = 0 (elements)
Memory per array = 7629.4 MiB (= 7.5 GiB).
Total memory required = 22888.2 MiB (= 22.4 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 112
Number of Threads counted = 112
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 34332 microseconds.
   (= 34332 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:          452770.4     0.035386     0.035338     0.035415
Scale:         450361.7     0.035659     0.035527     0.035943
Add:           467717.7     0.051403     0.051313     0.051461
Triad:         468265.3     0.051340     0.051253     0.051725
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three
   arrays
-------------------------------------------------------------
```

Avg time
0.035386
0.035659
0.051403
0.051340

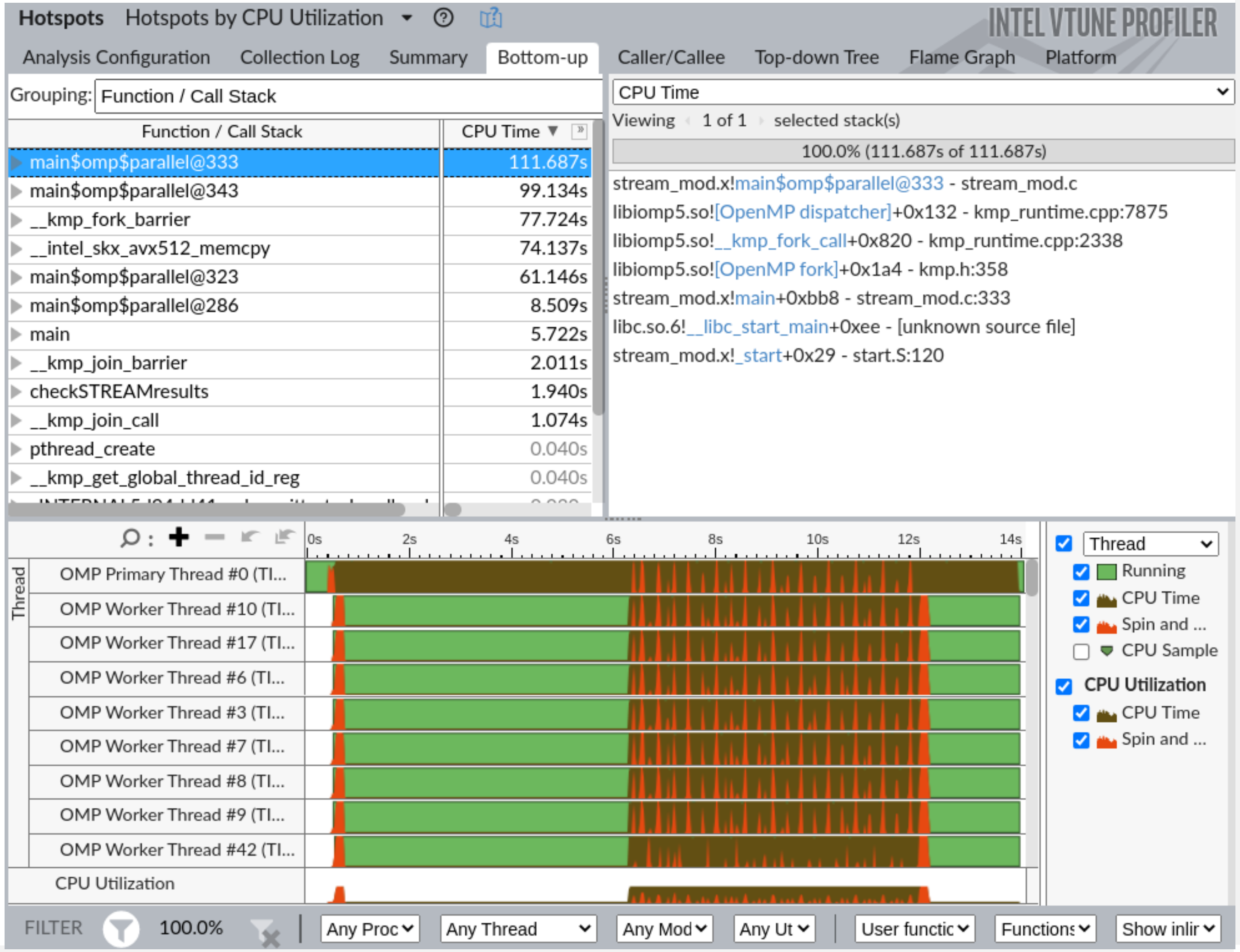
```
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 1000000000 (elements), Offset = 0 (elements)
Memory per array = 7629.4 MiB (= 7.5 GiB).
Total memory required = 22888.2 MiB (= 22.4 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 112
Number of Threads counted = 112
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 96448 microseconds.
   (= 96448 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:          275767.0     0.061914     0.058020     0.075499
Scale:         147795.1     0.108587     0.108258     0.109004
Add:           190052.3     0.129685     0.126281     0.138677
Triad:         204746.7     0.119864     0.117218     0.129618
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three
   arrays
-------------------------------------------------------------
```

Avg time
0.061914
0.108587
0.129685
0.119864

STREAM – VTune Hotspot SW Analysis

$ vtune -c hotspots (…)

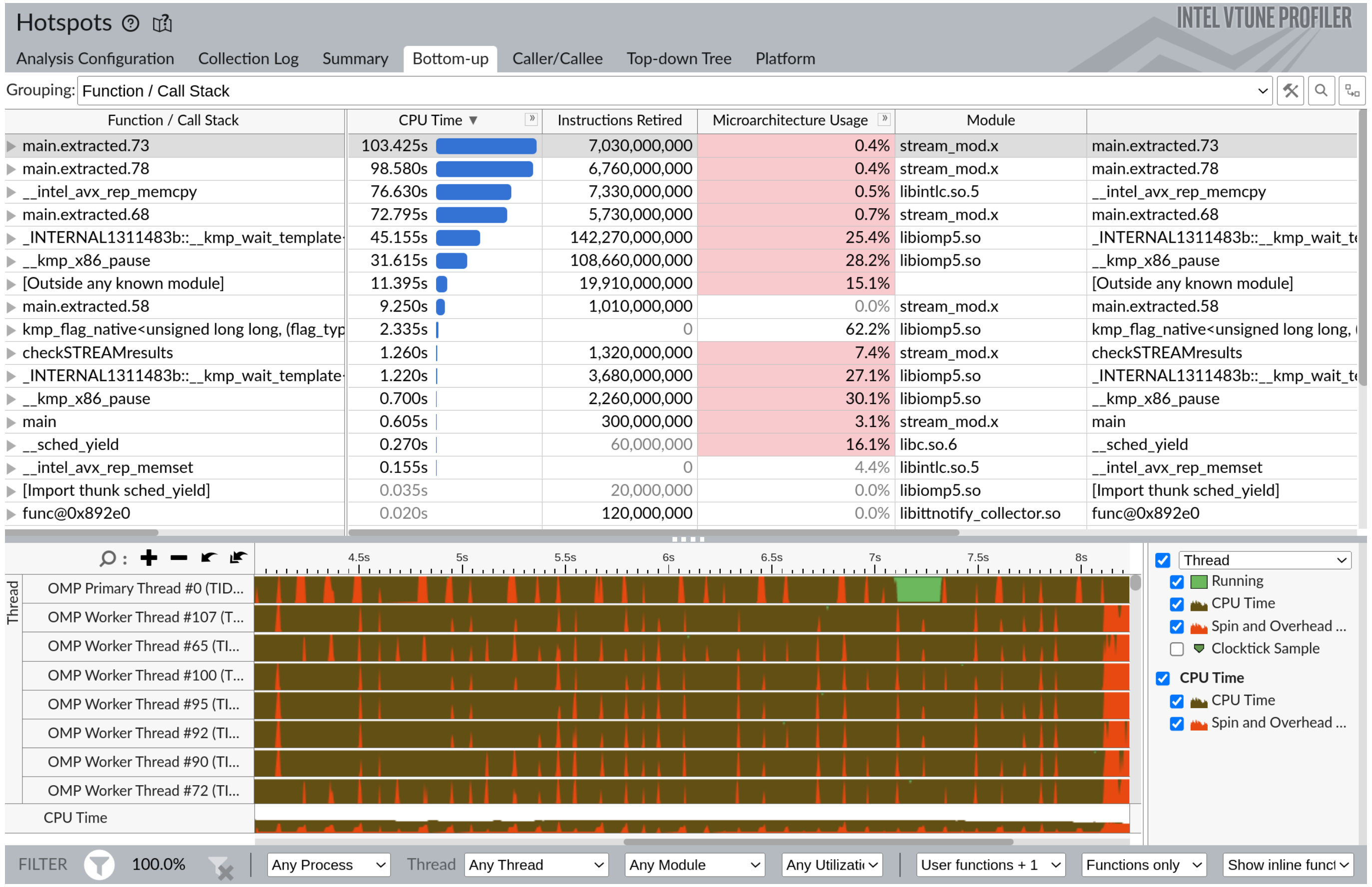
- Bottom-up ➟ Function/Call Stack: kernels & **func.** sorted by CPU Time
- Right-click **func.** ➟ filter by selection
- Clear all filters button (bottom)
- Select region ➟ Zoom & Filter by Selection
- Double-click **func.** **Gets redirected to source-code line + assembly**



STREAM – VTune Hotspot HW Analysis

$vtune -c hotspots \
  -knob sampling-mode=hw (…)

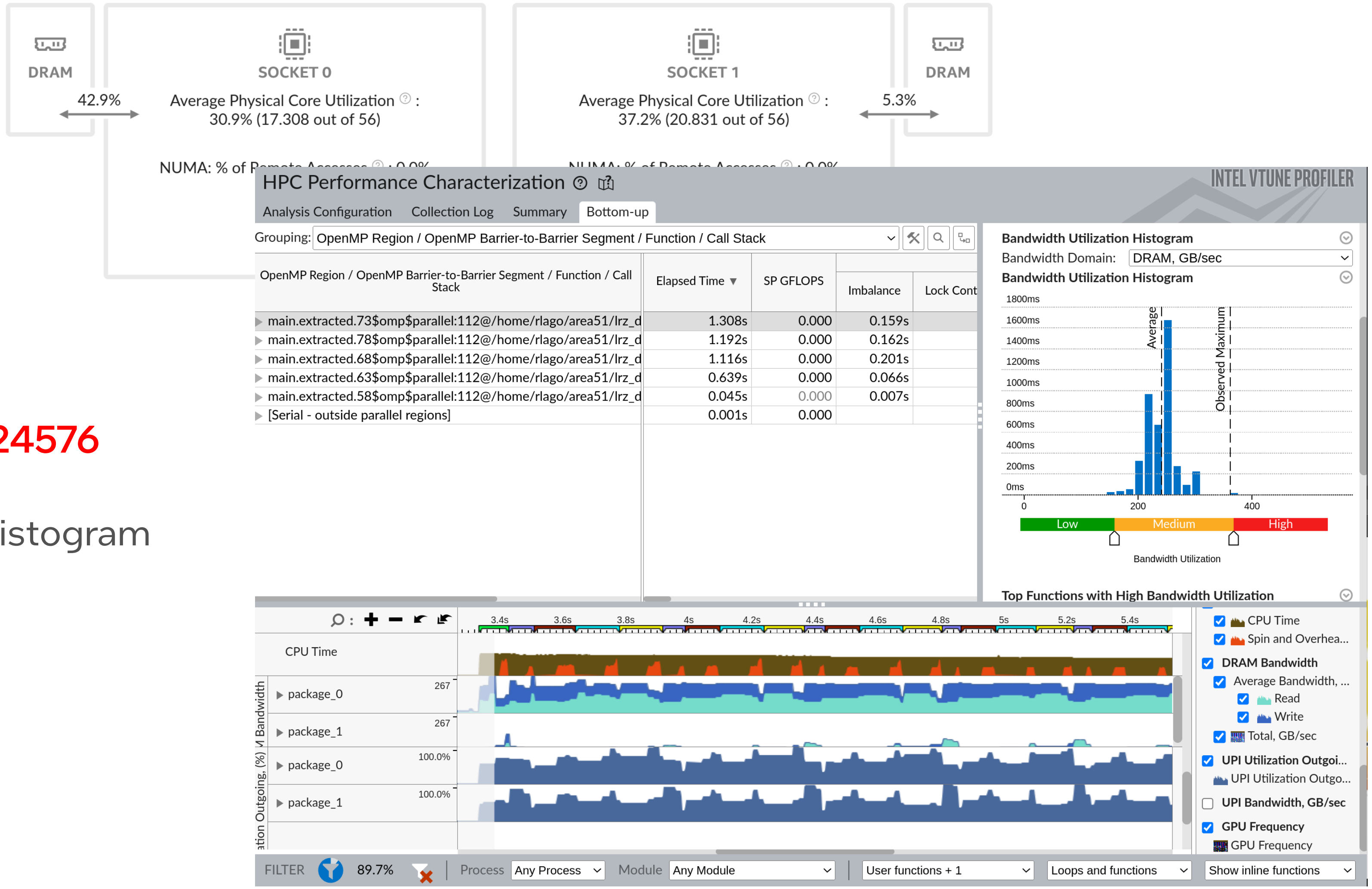
- New Columns:
  - Instructions Retired
  - µarch usage
- New Grouping:
  Physical Core/ Logical Core / Function / Call Stack
- New function:
  [Outside any known module] ➟ **Linux kernel!**



Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	Module	
main.extracted.73	103.425s	7,030,000,000	0.4%	stream_mod.x	main.extracted.73
main.extracted.78	98.580s	6,760,000,000	0.4%	stream_mod.x	main.extracted.78
__intel_avx_rep_memcpy	76.630s	7,330,000,000	0.5%	libintlc.so.5	__intel_avx_rep_memcpy
main.extracted.68	72.795s	5,730,000,000	0.7%	stream_mod.x	main.extracted.68
_INTERNAL1311483b::__kmp_wait_template	45.155s	142,270,000,000	25.4%	libiomp5.so	_INTERNAL1311483b::__kmp_wait_te
__kmp_x86_pause	31.615s	108,660,000,000	28.2%	libiomp5.so	__kmp_x86_pause
[Outside any known module]	11.395s	19,910,000,000	15.1%		[Outside any known module]
main.extracted.58	9.250s	1,010,000,000	0.0%	stream_mod.x	main.extracted.58
kmp_flag_native<unsigned long long, (flag_typ	2.335s	0	62.2%	libiomp5.so	kmp_flag_native<unsigned long long,
checkSTREAMresults	1.260s	1,320,000,000	7.4%	stream_mod.x	checkSTREAMresults
_INTERNAL1311483b::__kmp_wait_template	1.220s	3,680,000,000	27.1%	libiomp5.so	_INTERNAL1311483b::__kmp_wait_te
__kmp_x86_pause	0.700s	2,260,000,000	30.1%	libiomp5.so	__kmp_x86_pause
main	0.605s	300,000,000	3.1%	stream_mod.x	main
__sched_yield	0.270s	60,000,000	16.1%	libc.so.6	__sched_yield
__intel_avx_rep_memset	0.155s	0	4.4%	libintlc.so.5	__intel_avx_rep_memset
[Import thunk sched_yield]	0.035s	20,000,000	0.0%	libiomp5.so	[Import thunk sched_yield]
func@0x892e0	0.020s	120,000,000	0.0%	libittnotify_collector.so	func@0x892e0

STREAM – VTune HPC Analysis

vtune –c hpc-performance \
-analyze-system (…)



- **-analyze-system** requires **ulimit -n 24576**
- Memory Bound ➟ BW Utilization Histogram
- Bottom-up ➟ Zoom in region

STREAM – VTune Memory Analysis

$ vtune -c memory-access

- Summary shows poor use of L1-L3 cache
- Bottom-Up:
  - 1st socket: high bandwidth
  - 2nd socket: poor bandwidth (why?)
- Sort by LLC Miss Count
  - Arrays **a[]**, **b[]** and **c[]** are visible!

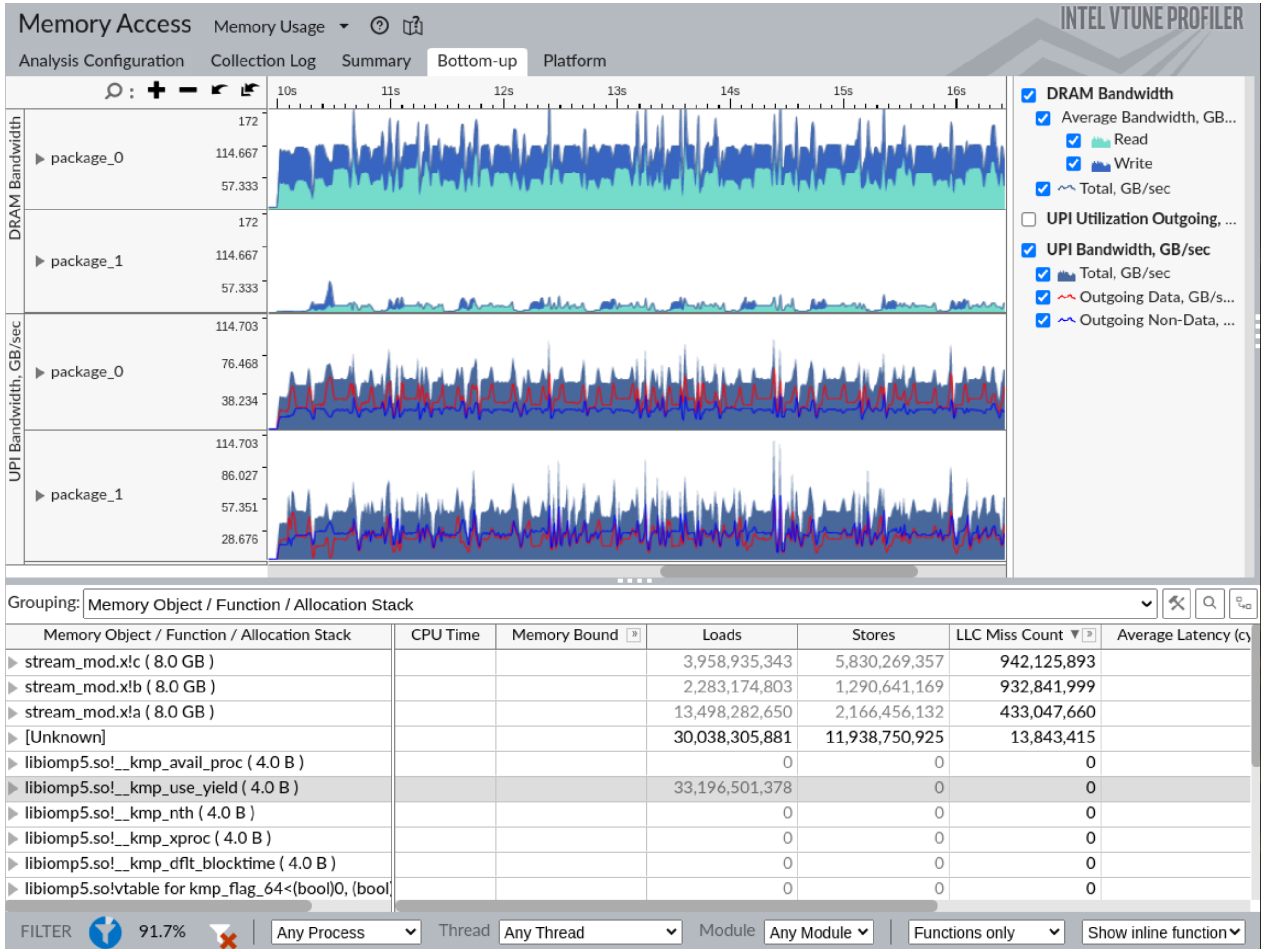
Elapsed Time: 9.070s

CPU Time:	431.310s	
Memory Bound:	**100.0%**	**of Pipeline Slots**
L1 Bound:	29.0%	of Clockticks
L2 Bound:	0.0%	of Clockticks
L3 Bound:	2.7%	of Clockticks
DRAM Bound:	**43.2%**	**of Clockticks**
DRAM Bandwidth Bound:	43.2%	of Elapsed Time
Store Bound:	0.9%	of Clockticks
NUMA: % of Remote Accesses:	0.0%	
UPI Utilization Bound:	28.2%	of Elapsed Time
Loads:	90,772,350,052	
Stores:	20,031,888,967	
LLC Miss Count:	**995,178,675**	
Total Thread Count:	112	
Paused Time:	0s	

STREAM – VTune Memory Analysis

$vtune -c memory-access \
 -knob analyze-mem-objects=true (...)

- Summary shows poor use of L1-L3 cache
- Bottom-Up:
  - 1st socket: high bandwidth
  - 2nd socket: poor bandwidth (why?)
- Sort by LLC Miss Count
  - Arrays a[], b[] and c[] are visible!



STREAM – Resolving the Cause

- Remove filter/zoom
  - Select stream_mod.x!a -> list of functions accessing it
  - Double-click **main** and find problem in source-code!
  - Line 267 was erroneously commented!
- Linux first touch policy
  - Memory is assigned to **NUMA domains when being touched** by the first time
  - Here, all memory is initialized in the master thread!

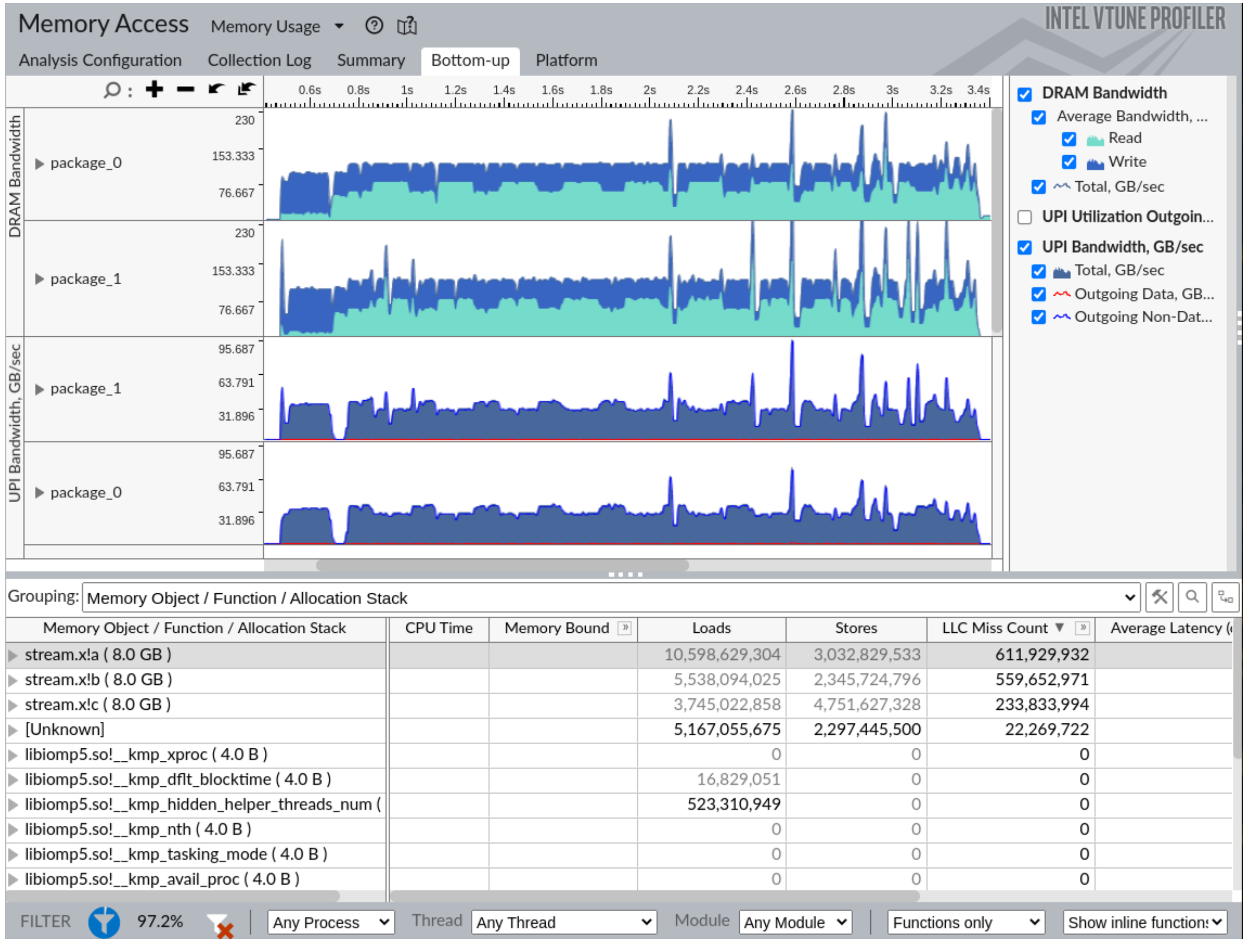
Source | Assembly

Source Line ▲	Source	CPU Time: Total	CPU Time: Self	Memory I

```
251            {
252                k = omp_get_num_threads();
253                printf ("Number of Threads requested = %i\n",k);
254            }
255        }
256    #endif
257
258    #ifdef _OPENMP
259            k = 0;
260    #pragma omp parallel
261    #pragma omp atomic
262                    k++;
263        printf ("Number of Threads counted = %i\n",k);
264    #endif
265
266        /* Get initial value for system clock. */
267    //#pragma omp parallel for
268        for (j=0; j<STREAM_ARRAY_SIZE; j++) {
269                a[j] = 1.0;
270                b[j] = 2.0;
271                c[j] = 0.0;
272            }
273
274        printf(HLINE);
275
276        if  ( (quantum = checktick()) >= 1)
277            printf("Your clock granularity/precision appears to be "
278                "%d microseconds.\n", quantum);
279        else {
280            printf("Your clock granularity appears to be "
281                "less than one microsecond.\n");
282            quantum = 1;
```

STREAM – Resolving the Cause

- Remove filter/zoom
  - Select stream_mod.x!a -> list of functions accessing it
  - Double-click **main** and find problem in source-code!
  - Line 267 was erroneously commented!
- Linux first touch policy
  - Memory is assigned to **NUMA domains when being touched** by the first time
  - Here, all memory is initialized in the master thread!



STREAM – Resolving the Cause

Elapsed Time: 13.490s

- CPU Time: 412.963s
  - Effective Time: 349.344s
  - Spin Time: 63.609s
  - Overhead Time: 0.010s
- Instructions Retired: 178,152,000,000
- Microarchitecture Usage: 4.0% of Pipeline Slots
  - CPI Rate: 5.333
- Total Thread Count: 73
- 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.

Function	Module	CPU Time	% of CPU Time
mainompparallel@333	stream_mod.x	105.733s	25.6%
mainompparallel@343	stream_mod.x	91.760s	22.2%
__intel_skx_avx512_memcpy	libintlc.so.5	68.045s	16.5%
mainompparallel@323	stream_mod.x	57.319s	13.9%
_INTERNAL92a63c0c::__kmp_wait_template<kmp_flag_64<(bool)0, (bool)1>, (bool)1, (bool)0, (bool)1>	libiomp5.so	37.017s	9.0%
[Others]	N/A*	53.089s	12.9%

Elapsed Time: 5.284s

- CPU Time: 214.262s
- Instructions Retired: 73,572,000,000
- Microarchitecture Usage: 3.9% of Pipeline Slots
  - CPI Rate: 6.908
- Total Thread Count: 73
- 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.

Function	Module	CPU Time	% of CPU Time
mainompparallel@333	stream.x	52.298s	24.4%
mainompparallel@343	stream.x	52.207s	24.4%
__intel_skx_avx512_memcpy	libintlc.so.5	35.233s	16.4%
mainompparallel@323	stream.x	34.616s	16.2%
[Outside any known module]	[Unknown]	10.740s	5.0%
[Others]	N/A*	29.168s	13.6%

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

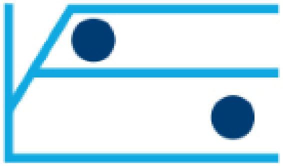
Intel® Advisor

Intel® Advisor

Rich Set of Capabilities for High Performance Code Design

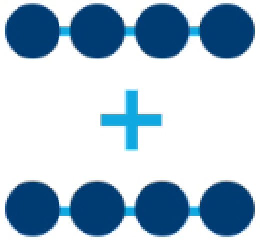
Offload Modelling

Design offload strategy and model performance on GPU.



Roofline Analysis

Optimize your application for memory and compute.

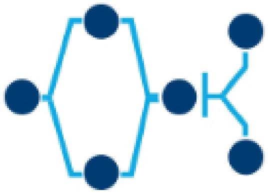


Vectorization Optimization

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.

“Automatic” Vectorization Often Not Enough

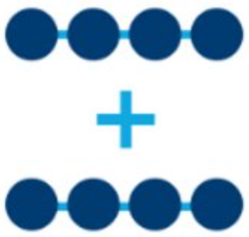
A good compiler can still benefit greatly from vectorization optimization

Compiler will not always vectorize

- Check for Loop Carried Dependencies
- Force vectorization:
  pragma simd (C++) or SIMD directive (Fortran)

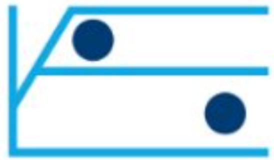
Not all vectorization is efficient vectorization

- Stride of 1 is more cache efficient than stride of 2 and greater
- Consider data layout changes
  Intel® SIMD Data Layout Templates can help



Vectorization Optimization

Enable more vector parallelism and improve its efficiency



Roofline Analysis

Optimize your application for memory and compute.

Advisor – CLI Essentials

- **Basic data collection:**
  action: --collect=survey

- **Tripcounts data collection:**
  action: --collect=tripcounts –flop

advisor <action> \
–project-dir=<project_path> \
-- <executable>

After data collection:

- **Roofline report:**
  action: --report=roofline –-report-output=<path_to_file>.html

- **Snapshot:**
  action: --snapshot --pack --cache-sources --cache-binaries ./snapshot

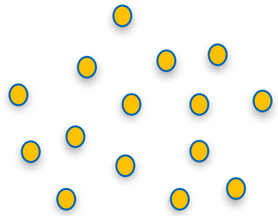
Demo 4 - NBody

```
GSimulation.cpp:
...
for (i = 0; i < n; i++) { // update acceleration
   for (j = 0; j < n; j++) {
        real_type distance, dx, dy, dz;
        real_type distanceSqr = 0.0;
        real_type distanceInv = 0.0;

        dx = particles[j].pos[0] - particles[i].pos[0];
        dy = particles[j].pos[1] - particles[i].pos[1];
        dz = particles[j].pos[2] - particles[i].pos[2];

        distSqr = dx*dx + dy*dy + dz*dz + softeningSquared;
        distInv = 1.0 / sqrt(distanceSqr);
        particles[i].acc[0] += dx * G * particles[j].mass * distInv * c
distInv;
        particles[i].acc[1] += …
        particles[i].acc[2] += …
   }
}
```

```
struct Particle {
  public:
    Particle() { init();}
    void init() {
      pos[0] = 0.; pos[1] = 0.; pos[2] = 0.;
      vel[0] = 0.; vel[1] = 0.; vel[2] = 0.;
      acc[0] = 0.; acc[1] = 0.; acc[2] = 0.;
      mass   = 0.;
    }
    real_type pos[3];
    real_type vel[3];
    real_type acc[3];
    real_type mass;
};
```



$$\vec{F}_{ij} = \frac{G\, m_i m_j}{|\vec{r}_j - \vec{r}_i|^3}(\vec{r}_j - \vec{r}_i)$$

$$\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = m\frac{d^2\vec{x}}{dt^2}$$

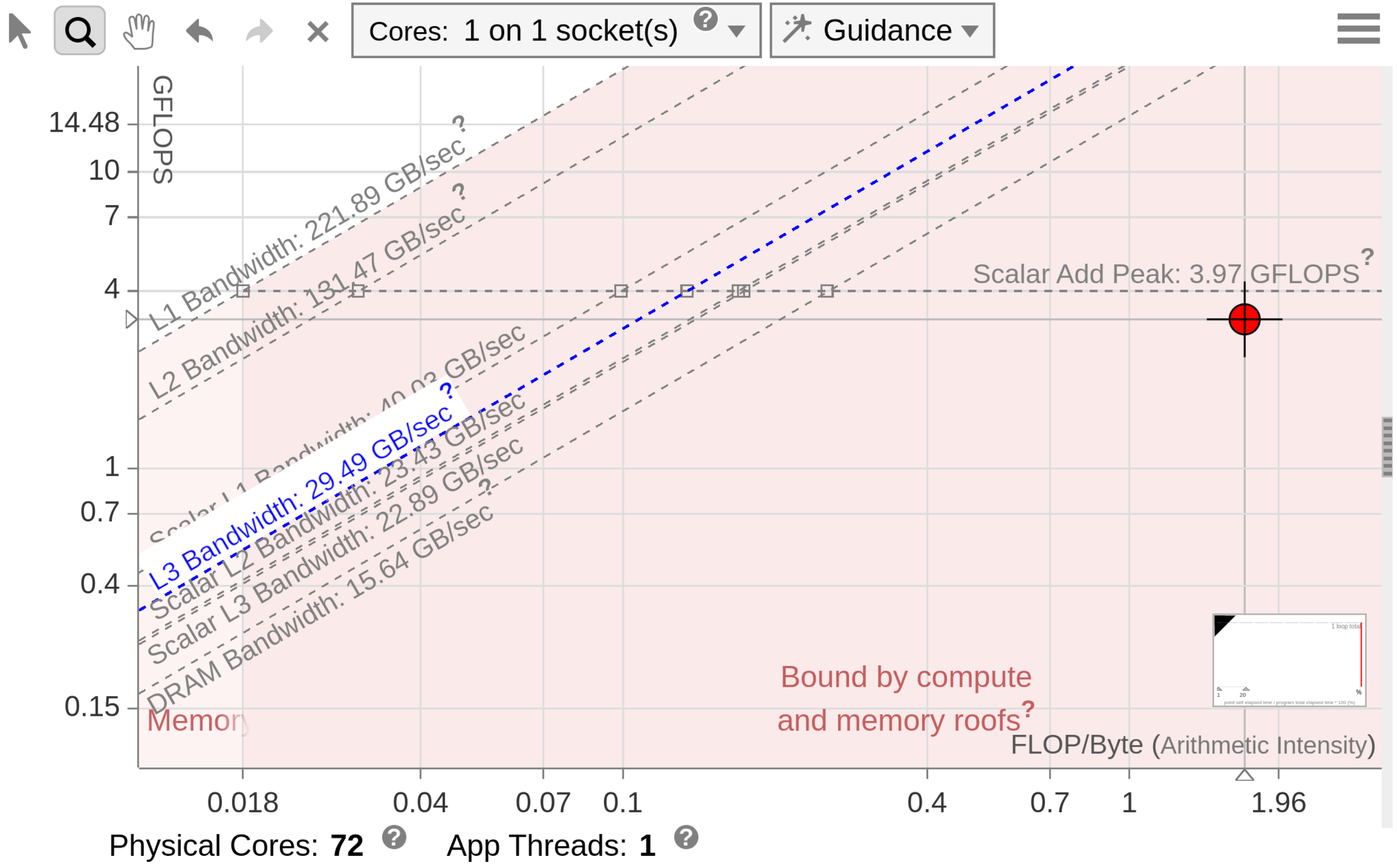
Pure C++; no MPI, SYCL nor OpenMP

Compiling & Running NBody Demo

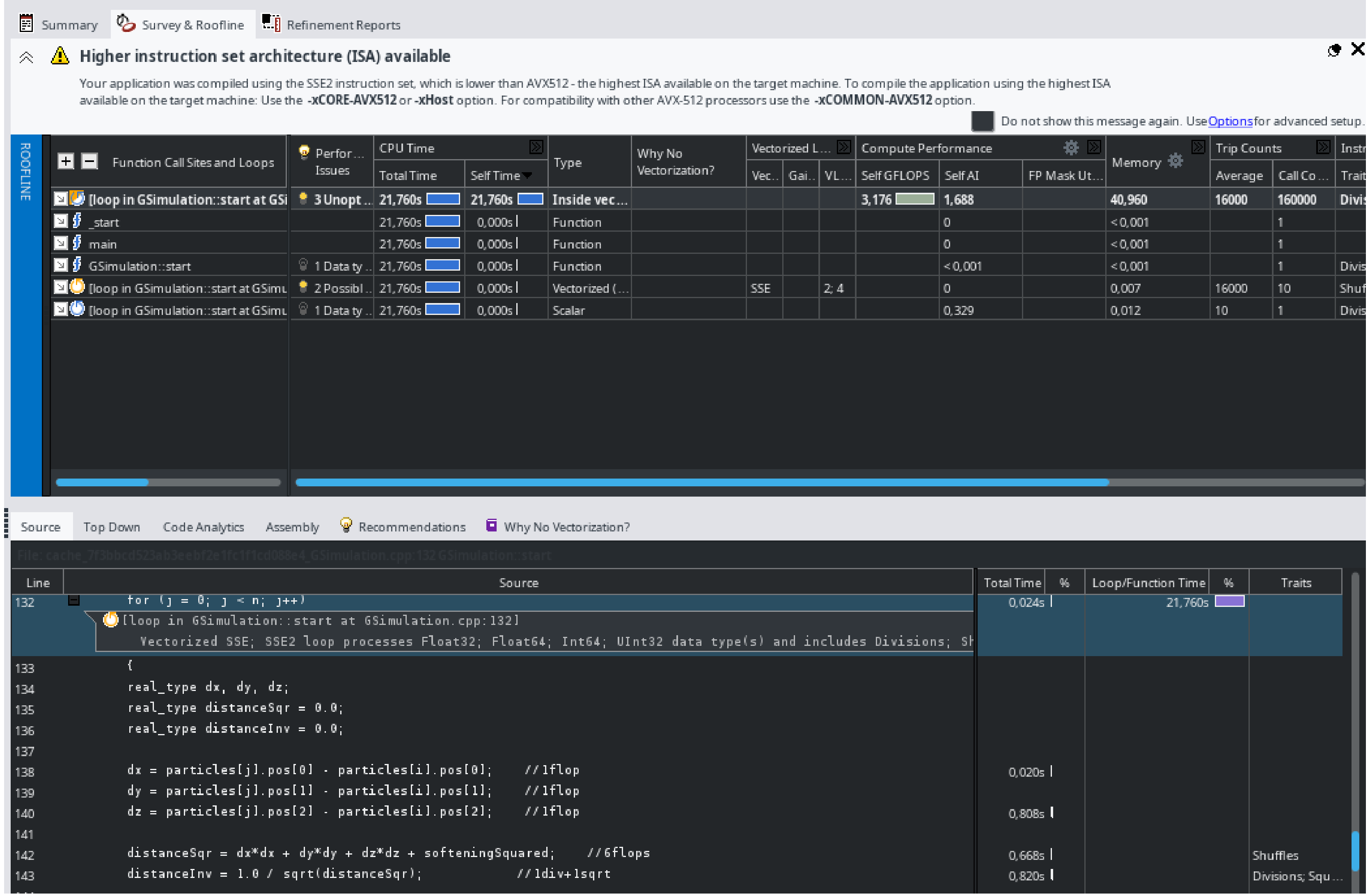
```
git clone  http://github.com/fbaru-dev/nbody-demo.git
module load oneapi/2023.1
cd ver0/
make CXX=icpx
make run
make survey
make roofline
advisor --report=roofline --project-dir=./adv-ver0 \
  --report-output=./ver0_roofline.html
advisor --snapshot --pack --cache-sources --cache-binaries \
  --project-dir=./adv-ver0 ./snapshot
```

NBody Demo – HTML Roofline Model

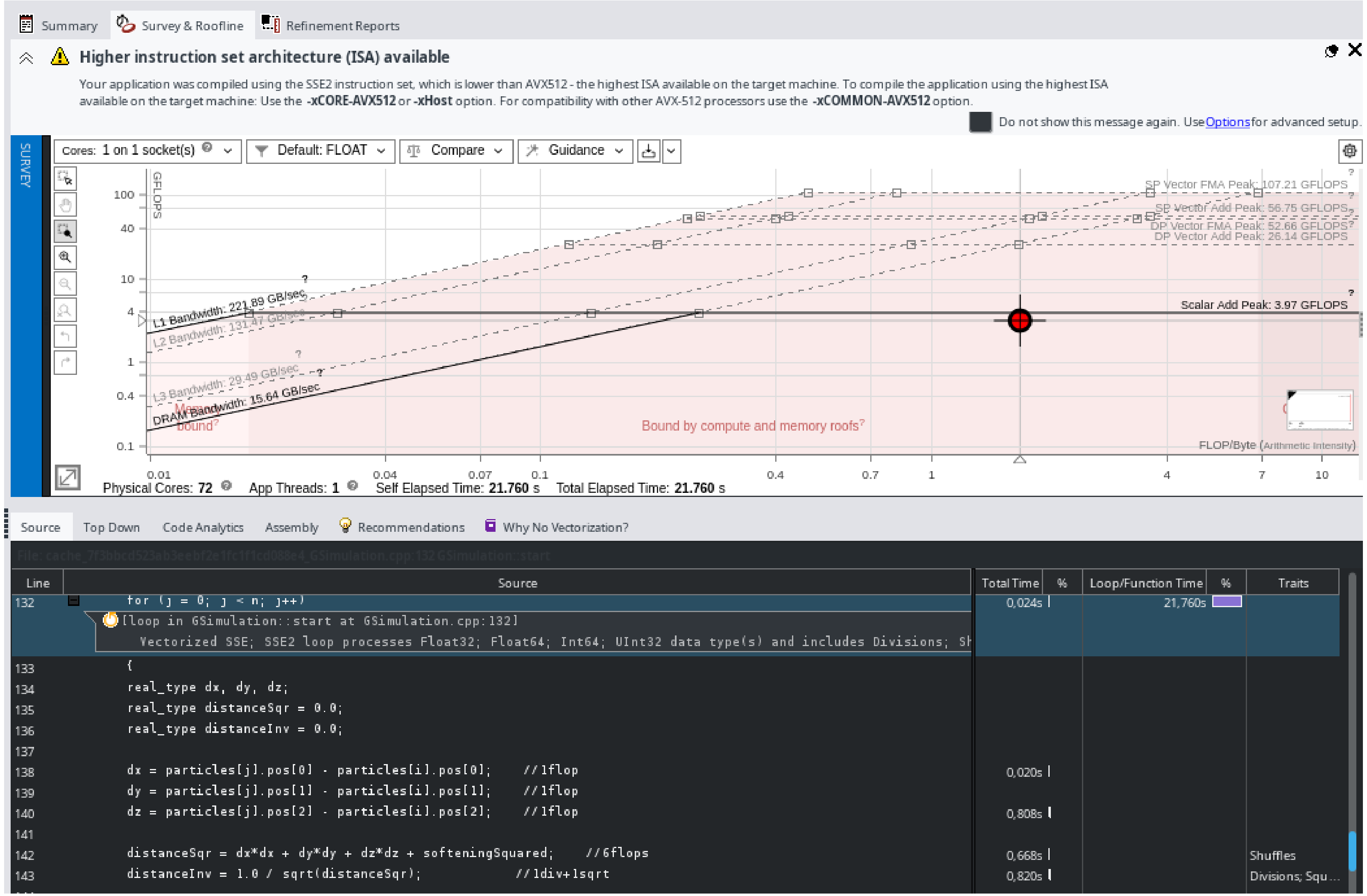
Attention: data collection may increase runtime up to 100x!



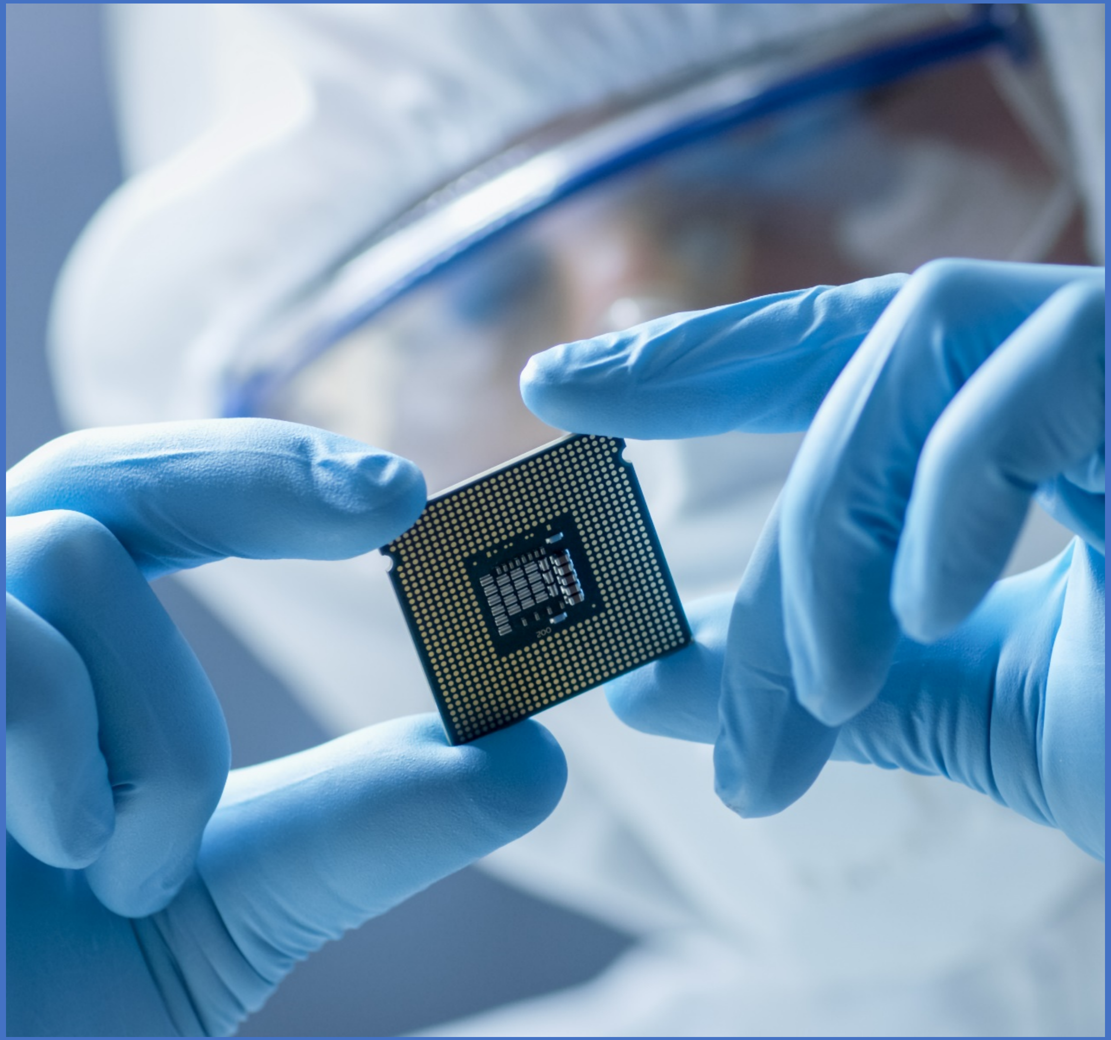
NBody Demo – Snapshot



NBody Demo – Snapshot



Questions?

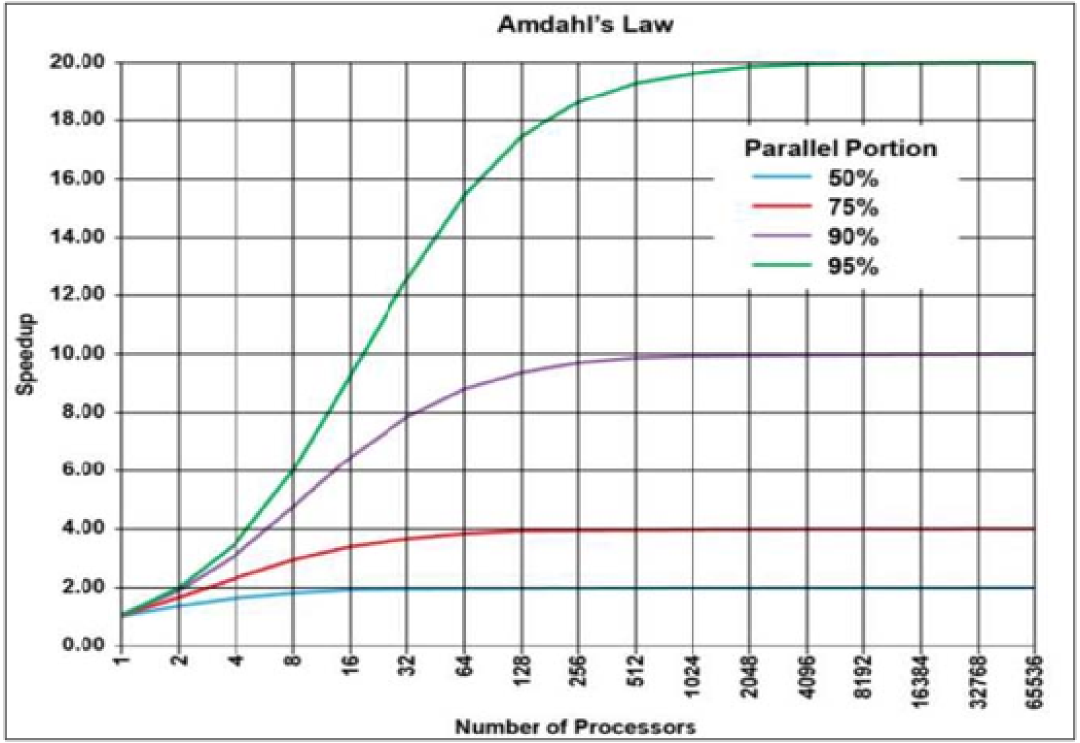


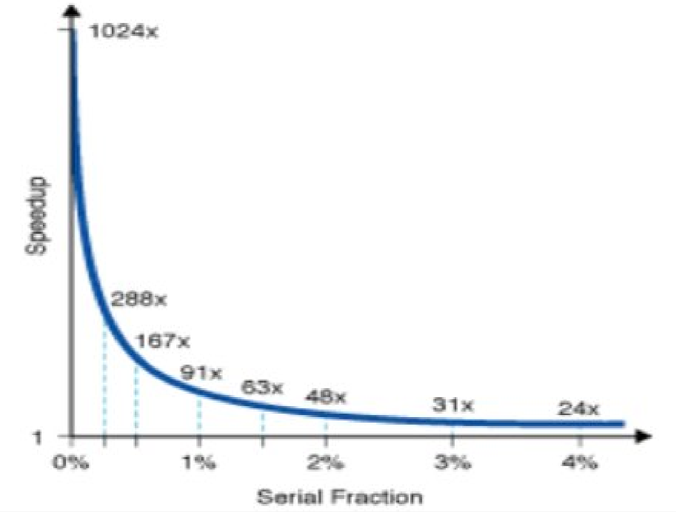
Amdahl’s Law

“The speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program.”

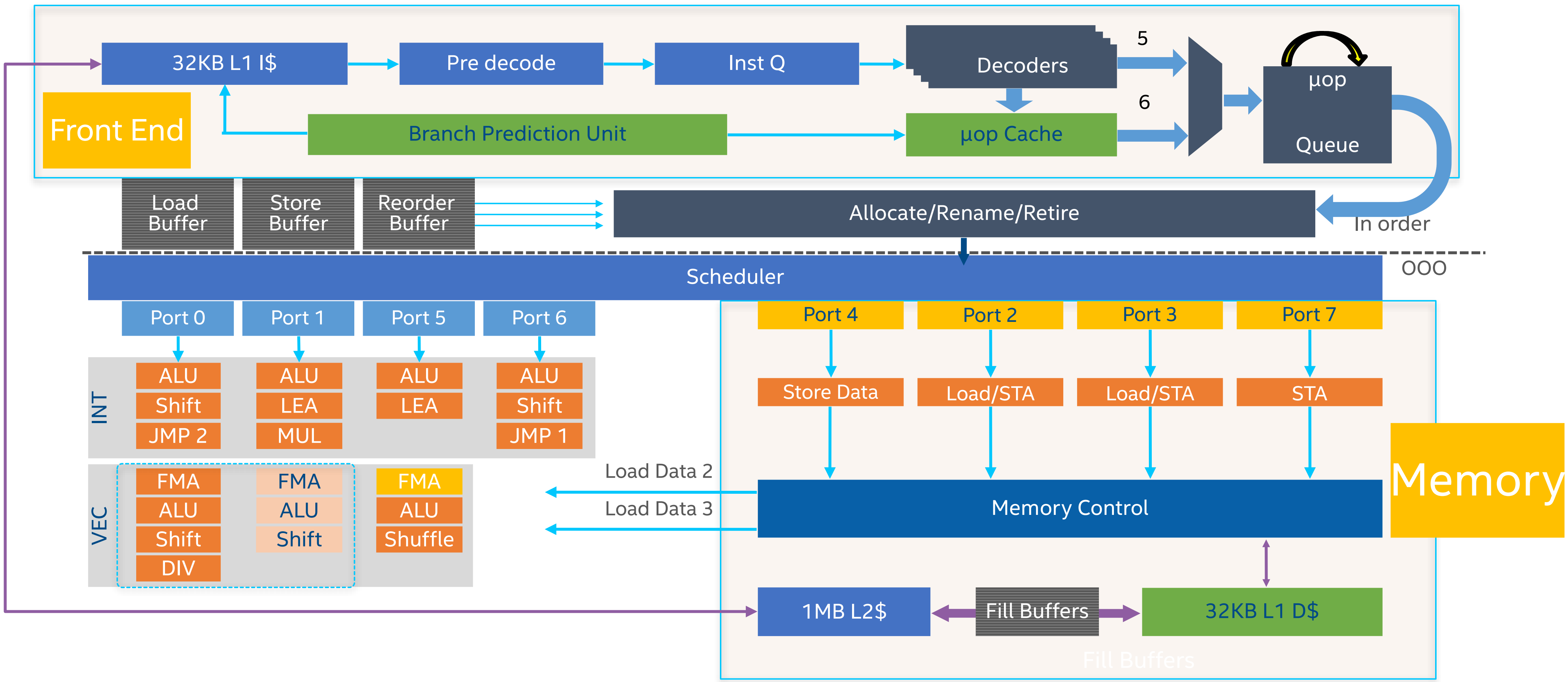
- Gene Amdahl

$$Speedup = (s + p) / (s + p / N)$$
$$= 1 / (s + p / N)$$





Instruction-level Parallelism (ILP)



Collection

How to collect:

mpirun [mpi_options] **aps [aps_options]** <app> [app_options]

Adjustable collection:

- --collection-mode=[mpi|omp|hwc|all] – ‘all’ by default
- --stat-level=[1..5] – from timing to detailed info about message sizes, communicators, destinations.
- --mpi-imbalance=[0..2] – 0 – disabled, 1 – get imbalance from Intel MPI (default), 2 – using inserted barriers
- Collection control through MPI_Pcontrols and ITT API

Low overhead:

- ~ 1-2% in default mode
- < 10% in any other mode

GPU metrics

- GPU execution efficiency
  - OA HW counters (per node)
- OpenMP offload efficiency
  - tracing through OMPT (per rank)

```
[root@nntpat98-144 aps_results]# aps --report --metrics="GPU Time"  ./aps_result_with_pci/
Loading 100.00%
| Metric Table
|-----------------------------------------
Metric Name          Node Name   Metric Value
GPU Time, s          s011-n004          1.307
GPU Time, s          s011-n005          0.004
[root@nntpat98-144 aps_results]# aps --report --metrics="GPU Time (% of Elapsed Time)"  ./aps_result_with_pci/
Loading 100.00%
| Metric Table
|---------------------------------------------------------------------------
Metric Name                                       Node Name   Metric Value
GPU Time (% of Elapsed Time), % of Elapsed Time   s011-n004           19.5
GPU Time (% of Elapsed Time), % of Elapsed Time   s011-n005            0.1
[root@nntpat98-144 aps_results]# aps --report --metrics="GPU Time (% of Elapsed Time)","GPU Utilization when Bu
Loading 100.00%
| Metric Table
|---------------------------------------------------------------------------
Metric Name                                       Node Name   Metric Value
GPU Time (% of Elapsed Time), % of Elapsed Time   s011-n004           19.5
GPU Time (% of Elapsed Time), % of Elapsed Time   s011-n005            0.1
GPU Utilization when Busy, %                      s011-n004           21.9
GPU Utilization when Busy, %                      s011-n005              0
GPU Occupancy, % of Peak Value                    s011-n004           84.4
GPU Occupancy, % of Peak Value                    s011-n005              0
```

GPU Utilization when Busy

10.95%

EU State	% of EUs
Active	10.95%
Idle	54.7%
Stalled	34.4%

Offload Activity	% of GPU time
Compute	36.31%
Overhead	5.1%
Data Transfer	58.59%

GPU Occupancy

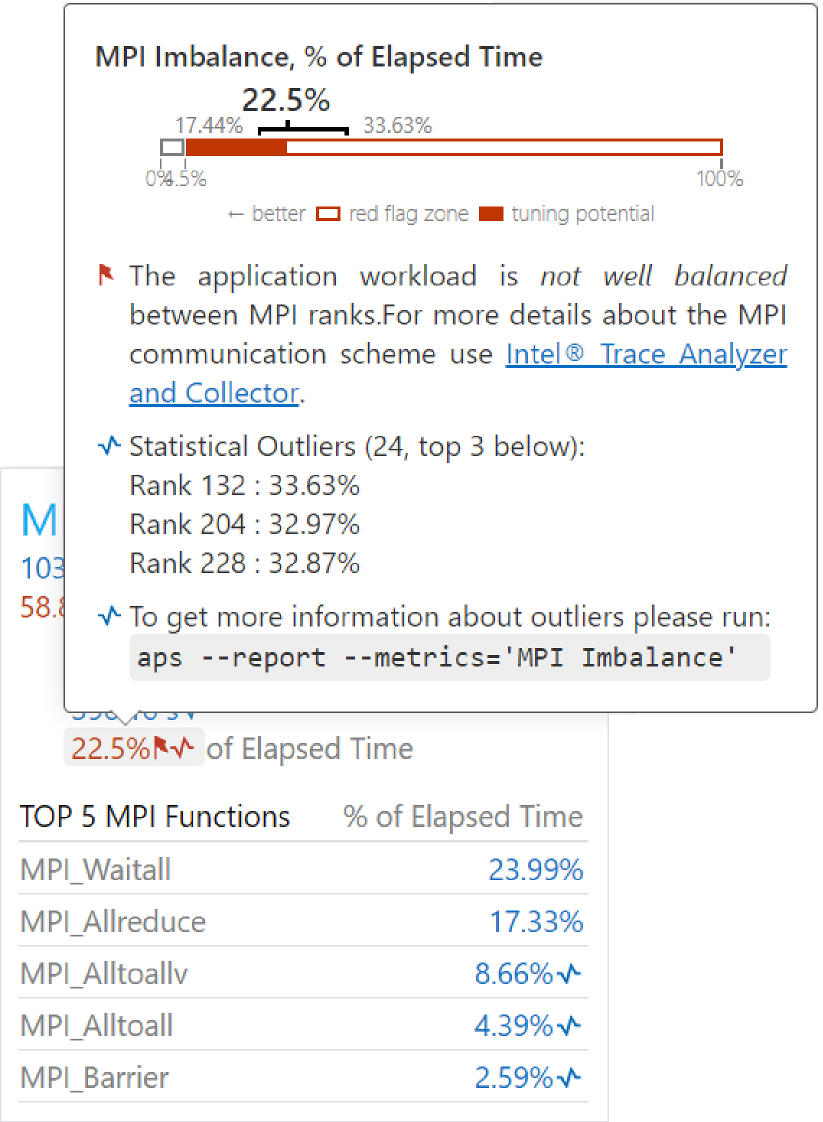
42.2% of Peak Value

Outliers

Provide Min, Max, Average

Detect statistical and threshold outliers

- Statistical outlier is based on two-sided Grubbs's test with 0.05 significance level
  - Highlighting anomalies and asymmetric distribution of work
  - Show a potential target for detailed analysis
- Threshold outlier – a metric value breaking the threshold.
  - Show an additional tuning potential for a source breaking the threshold.



Autotuner Example

Configuration possibly slowing down tuning run in favour of results.:

- I_MPI_TUNING_MODE=auto
- I_MPI_TUNING_AUTO_WARMUP_ITER_NUM=1
- I_MPI_TUNING_AUTO_ITER_NUM=128
- I_MPI_TUNING_AUTO_SYNC=1
- I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD=4194304
- I_MPI_TUNING_AUTO_STORAGE_SIZE=4194304
- I_MPI_TUNING_BIN_DUMP=./my_tuning_file.dat

Apply tuning results via

- I_MPI_TUNING_BIN=./my_tuning_file.dat

Restricting the scope of implementations

Remove failed implementation/s and switch back to the release version of Intel MPI Library and rerun autotuner. E.g. removing 11th implementation.:

```
$ export I_MPI_ADJUST_ALLREDUCE_LIST=0-10,12-25
```

This technique can also be used outside of tuning scenarios to find failed implementations in Intel MPI Library.

mpitune_fast

	Autotuner	mpitune_fast
Scope	**Application specific tuning**	**Cluster wide tuning**
Intended for	**Regular users**	**System administrators**

- tunes the Intel® MPI Library to the cluster configuration using autotuner functionality.
- iteratively launches the Intel® MPI Benchmarks with the proper autotuner environment and generates a tuning file.
- supports Slurm and LSF job managers. mpitune_fast automatically finds job allocated hosts and performs launches.
- **Example**

```
$ mpitune_fast -f ./hostfile -c alltoall,allreduce,barrier
```

VTune - Add Custom Counters to the Timeline

Import a file or use the new API

Visualize your software counters on the timeline

- E.g.: Frames/second, packets/second, matrix operations/second
- Quickly see what code is executing when your counters change

Example: Create a counter for temperature and memory usage metrics.

```
#include "ittnotify.h"
__itt_counter temperatureCounter = __itt_counter_create("Temperature", "Domain");
__itt_counter memoryUsageCounter = __itt_counter_create("Memory Usage", "Domain");
unsigned __int64 temperature;
while (...)
temperature = getTemperature();
__itt_counter_set_value(temperatureCounter, &temperature);
__itt_counter_inc_delta(memoryUsageCounter, getAllocatedMemSize());
__itt_counter_dec_delta(memoryUsageCounter, getDeallocatedMemSize());
...
}
__itt_counter_destroy(temperatureCounter);
__itt_counter_destroy(memoryUsageCounter);
```