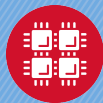


Introduction to Performance Tools

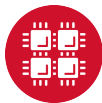
Samuel Khuvis

Scientific Applications Engineer, OSC



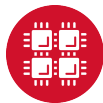
Goals of the Breakout Session

- ▶ Let you know what tools are available at OSC
- ▶ Suggest when you should use each of them
- ▶ Give an overview of usage for each
 - ▶ Including a demo or sample output
- ▶ Show you where to find more information



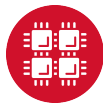
For More Information

- ▶ Visit the software pages on our website
www.osc.edu
Resource → Available Software
- ▶ Contact the help desk (OSC Help)
oschelp@osc.edu
614-292-1800
1-800-686-6472
- ▶ Optimization and Performance Tuning Workshop on October 29, 2019 at 1-4 pm
osc.edu/calendar/events/2019_10_29-optimization_performance_tuning_workshop
- ▶ Self-guided tutorial: https://www.osc.edu/resources/getting_started/howto/howto_tune_performance



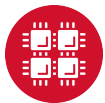
Profiling/Debugging Tools Available at OSC

- ▶ Parallel debugging tools
 - ▶ ARM DDT
- ▶ Profiling tools
 - ▶ ARM Performance Reports
 - ▶ ARM MAP
 - ▶ Intel VTune
 - ▶ Intel Trace Analyzer and Collector (ITAC)
 - ▶ Intel Advisor
 - ▶ TAU Commander
 - ▶ HPCToolkit



What can a debugger do for you?

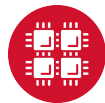
- ▶ Debuggers let you
 - ▶ execute your program one line at a time (“step”)
 - ▶ inspect variable values
 - ▶ stop your program at a particular line (“breakpoint”)
 - ▶ open a “core” file (after program crashes)
- ▶ HPC debuggers
 - ▶ support multithreaded code
 - ▶ support MPI code
 - ▶ support GPU code
 - ▶ provide a nice GUI



Compilation flags for debugging

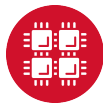
For debugging:

- ▶ Use `-g` flag
- ▶ Remove optimization or set to `-O0`
- ▶ Examples:
 - ▶ `icc -g -o mycode mycode.c`
 - ▶ `gcc -g -O0 -o mycode mycode.c`
- ▶ Use `icc -help diag` to see what compiler warnings and diagnostic options are available for the Intel compiler
- ▶ Diagnostic options can also be found by reading the man page of `gcc` with `man gcc`



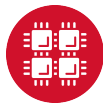
ARM DDT

- ▶ Available on all OSC clusters
 - ▶ `module load arm-ddt`
- ▶ To run a non-MPI program from the command line:
 - ▶ `ddt --offline --no-mpi ./mycode [args]`
- ▶ To run a MPI program from the command line:
 - ▶ `ddt --offline -np num_procs ./mycode [args]`



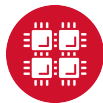
ARM DDT GUI

- ▶ To run ARM DDT as a GUI, login to OnDemand at `ondemand.osc.edu`
- ▶ To get an interactive session on a compute node, select “Pitzer Desktop” under “Interactive Apps”
- ▶ Enter information and click “Launch”
- ▶ Click “Launch noVNC in New Tab” to launch the desktop in a new tab
- ▶ From there you can open a terminal and run DDT as a GUI
- ▶ For a non-MPI program:
 - ▶ `ddt --no-mpi ./mycode [args]`
- ▶ For a MPI program:
 - ▶ `ddt -np num_procs ./mycode [args]`
- ▶ More information on using OnDemand is available at `osc.edu/resources/online_portals/ondemand`



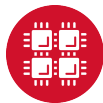
What can a profiler show you?

- ▶ Whether code is
 - ▶ compute-bound
 - ▶ memory-bound
 - ▶ communication-bound
- ▶ How well the code uses available resources
 - ▶ Multiple cores
 - ▶ Vectorization
- ▶ How much time is spent in different parts of the code



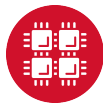
Compilation flags for profiling

- ▶ For profiling
 - ▶ Use `-g` flag
 - ▶ Explicitly specify optimization level `-O0`
 - ▶ Example: `icc -g -O0 -o mycode mycode.c`
- ▶ Use the same level optimization as you normally do
 - ▶ Bad example: `icc -g -o mycode mycode.c`
 - ▶ Equivalent to `-O0`



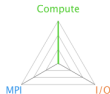
ARM Performance Reports

- ▶ Easy to use
 - ▶ “-g” flag not needed - works on precompiled binaries
- ▶ Gives a summary of your code's performance
 - ▶ view report with browser
- ▶ For a non-MPI program:
 - ▶ `module load arm-pr`
 - ▶ `perf-report --no-mpi ./mycode [args]`
- ▶ For an MPI program:
 - ▶ `perf-report -np num_procs ./mycode [args]`



arm
PERFORMANCE
REPORTS

```
Command: /fs/project/PZS0720/skhuis/SETSM/setsm
dataset/WV01_15MAY080613301-
P1B5-102001003C02A600.tif
dataset/WV01_15MAY080614188-
P1B5-102001003EA5DA00.tif out -outr5 8
-projection ps
Resources: 1 node (40 physical, 40 logical cores per node)
Tasks: 1 process, OMP_NUM_THREADS was 28
Machine: p0165.ten.osc.edu
Start time: Fri Dec 28 2018 14:13:20 (UTC-05)
Total time: 372 seconds (about 6 minutes)
Full path: /fs/project/PZS0720/skhuis/SETSM
```



Summary: setsm is **Compute-bound** in this configuration

Compute	99.3%	<div style="width: 99.3%; height: 15px; background-color: #00b050;"></div>	Time spent running application code. High values are usually good. This is very high ; check the CPU performance section for advice
MPI	0.0%	<div style="width: 0.0%; height: 15px; background-color: #0070c0;"></div>	Time spent in MPI calls. High values are usually bad. This is very low ; this code may benefit from a higher process count
I/O	0.7%	<div style="width: 0.7%; height: 15px; background-color: #e69d00;"></div>	Time spent in filesystem I/O. High values are usually bad. This is very low ; however single-process I/O may cause MPI wait times

This application run was **Compute-bound**. A breakdown of this time and advice for investigating further is in the **CPU** section below.

As very little time is spent in **MPI** calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the **99.3%** CPU time:

Single-core code	44.5%	<div style="width: 44.5%; height: 10px; background-color: #0070c0;"></div>
OpenMP regions	55.5%	<div style="width: 55.5%; height: 10px; background-color: #00b050;"></div>
Scalar numeric ops	21.8%	<div style="width: 21.8%; height: 10px; background-color: #0070c0;"></div>
Vector numeric ops	4.4%	<div style="width: 4.4%; height: 10px; background-color: #00b050;"></div>
Memory accesses	43.7%	<div style="width: 43.7%; height: 10px; background-color: #0070c0;"></div>

The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.

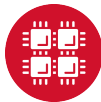
Little time is spent in **vectorized instructions**. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the **0.0%** MPI time:





Time in collective calls	0.0%	
Time in point-to-point calls	0.0%	
Effective process collective rate	0.00 bytes/s	
Effective process point-to-point rate	0.00 bytes/s	

No time is spent in **MPI** operations. There's nothing to optimize here!



I/O

A breakdown of the 0.7% I/O time:

Time in reads	71.4%	
Time in writes	28.6%	
Effective process read rate	2.88 GB/s	
Effective process write rate	3.23 GB/s	

Most of the time is spent in **read operations** with a **high effective transfer rate**. It may be possible to achieve faster effective transfer rates using asynchronous file operations.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	1.16 GiB	
Peak process memory usage	3.70 GiB	
Peak node memory usage	8.0%	

The **peak node memory usage** is very low. Larger problem sets can be run before scaling to multiple nodes.

OpenMP

A breakdown of the 55.5% time in OpenMP regions:

Computation	78.5%	
Synchronization	21.5%	
Physical core utilization	70.0%	
System load	57.9%	

OpenMP thread performance looks good. Check the CPU breakdown for advice on improving code efficiency.

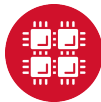
Energy

A breakdown of how the 19.1 Wh was used:

CPU	100.0%	
System	not supported %	
Mean node power	not supported W	
Peak node power	0.00 W	

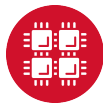
The **whole system energy** has been calculated using the **CPU energy usage**.

System power metrics: No Arm IPMI Energy Agent config file found in `/var/spool/ipmi-energy-agent`. Did you start the Arm IPMI Energy Agent?



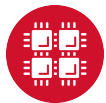
ARM MAP

- ▶ Interpretation of profile requires some expertise
- ▶ Gives details about your code's performance
- ▶ For a non-MPI program:
 - ▶ `module load arm-map`
 - ▶ `map --profile --no-mpi ./mycode [args]`
- ▶ For an MPI program:
 - ▶ `map --profile -np num_procs ./mycode [args]`
- ▶ View and explore resulting profile using ARM client
 - ▶ Download remote client to view profiles on local machine at developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge
 - ▶ Information on transferring files to your local machine at osc.edu/resources/online_portals/ondemand/file_transfer_and_management

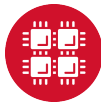
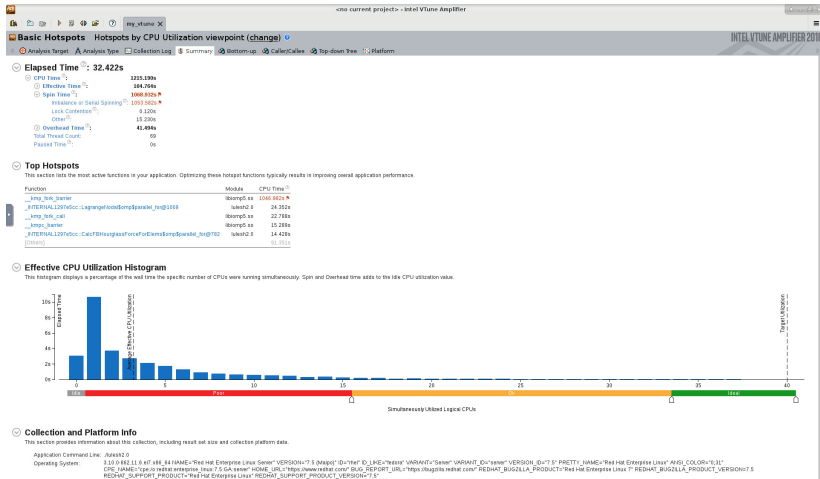


Intel VTune

- ▶ A profiler that can work with C, C++, Fortran programs
- ▶ Works best on a single node
- ▶ For using a GUI (use for small problems < 5 minutes):
 - ▶ `amplxe-gui`
- ▶ For non-interactive usage:
 - ▶ `amplxe-cl -r my_vtune -collect hotspots -no-auto-finalize ./mycode`
 - ▶ `amplxe-cl -report hotspots -r my_vtune`
- ▶ View and explore existing results with `amplxe-gui`

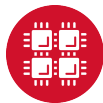


VTune GUI

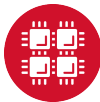
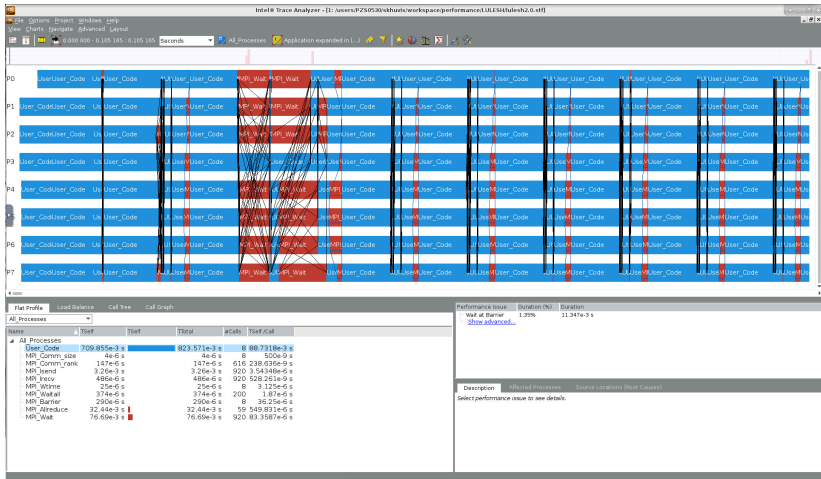


Intel Trace Analyzer and Collector (ITAC)

- ▶ Graphical tool for profiling MPI code (Intel MPI)
- ▶ To use:
 - ▶ `module load intelmpi # then compile (-g) code`
 - ▶ `mpiexec -trace ./mycode`
- ▶ View and explore existing results using GUI with `traceanalyzer`:
 - ▶ `traceanalyzer <mycode>.stf`

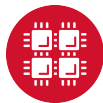


ITAC GUI



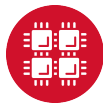
TAU Commander

- ▶ Tool that can be used to profile, trace, or sample your application
- ▶ Load with `taucmdr` module
- ▶ Requires moderate amount of setup:
 - ▶ Create a project with information about the application
 - ▶ For example, `tau initialize --mpi --compilers Intel`
 - ▶ Select appropriate measurement:
 - ▶ `tau select sample`
- ▶ To use:
 - ▶ `tau mpiexec ./mycode`
- ▶ To view and explore profile data:
 - ▶ `tau trial show <trial_number>`

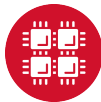
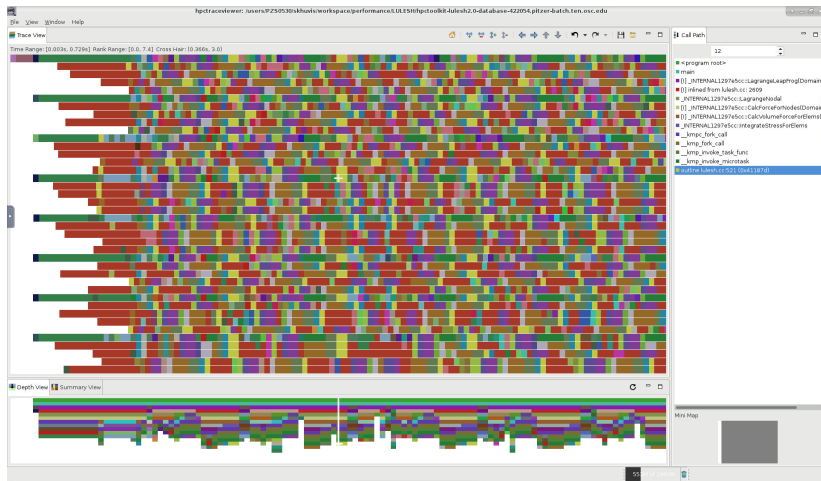


HPCToolkit

- ▶ Suite of tools that can be used to profile or trace your application
- ▶ Load with `hpctoolkit` module
- ▶ To profile your application: `mpiexec hpcrun ./mycode`
- ▶ This will produce a directory with a name of the form `hpctoolkit-mycode-measurements-pid.nodeid` containing profile data
- ▶ To convert the output to a format that can be viewed by the `hpcviewer` tool, run
`hpcprof hpctoolkit-mycode-measurements-pid.nodeid`
- ▶ To view the profile data generated during the run in a GUI, call `hpcviewer`
`hpctoolkit-mycode-database-pid.nodeid`

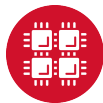


HPCToolkit Profile



Intel Advisor

- ▶ Graphical tool for optimizing vectorization and threading
- ▶ For using a GUI (use for small problems < 5 minutes):
 - ▶ `advixe-gui`
- ▶ For non-MPI non-interactive usage:
 - ▶ `advixe-cl -collect survey -project-dir ./my_advisor ./mycode`
- ▶ For MPI non-interactive usage:
 - ▶ `mpirun -n <mpi_tasks> advixe-cl -collect survey -project-dir ./my_advisor ./mycode`
- ▶ View and explore existing results with `advixe-gui`



users/PZ56520\khuhof\workspace\performance\UEI\HVM_advisor - Intel Advisor

File View Help Start Survey Analysis Rank: 0

Welcome Rank: 0

Vectorization Workflow Threading Workflow

Elapsed time: 83.97s Vectorized Not Vectorized FILTER All Modules All Sources

Summary Survey & Routine Refinement Reports

Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vector parallelism, discover performance issues preventing from effective vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Routine model automation.

Program metrics
Elapsed Time: 83.97s
Vector Instruction Set: AVX, SSE2, SSE Number of CPU Threads: 5

Loop metrics Metrics

Metric	Total
Total CPU time	295.82s 100.0%
Time in 2D vectorized loops	15.04s 5.1%
Time in scalar code including time in 3D vectorized completely unrolled loops	281.78s 94.9%

Vectorization Gains/Efficiency
Vectorized Loops Gain/Efficiency: 1.66x 8.9%
Program Approximate Gain: 1.03x

Per program recommendations
Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA. [Show more](#)

Top time-consuming loops

Loop	Self Time	Total Time
Loop in _NTERMINALL207fccc-Lapwarp@Node\komp@para\hw_for@1009 at lshsh.cc:1010	5.847s	5.847s
Loop in _NTERMINALL207fccc-CaB@MourgliaS@rce@SciEms@komp@para\hw_for@782 at lshsh.cc:795	3.450s	3.450s
Loop in _NTERMINALL207fccc-CaB@MourgliaS@rce@SciEms@komp@para\hw_for@1018 at lshsh.cc:1018	3.096s	3.096s
Loop in _NTERMINALL207fccc-CaB@MourgliaS@rce@SciEms@komp@para\hw_for@958 at lshsh.cc:877	2.670s	2.670s
Loop in _NTERMINALL207fccc-CaB@MourgliaS@rce@SciEms@komp@para\hw_for@782 at lshsh.cc:783	2.255s	9.501s

Recommendations

Loop	Self Time	Recommendations
Loop in _NTERMINALL207fccc-CaB@MourgliaS@rce@SciEms@komp@para\hw_for@782 at lshsh.cc:795	3.450s	Set loop_ana_ana\hw_for@782

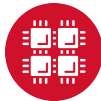
Collection details

Platform information

MPI rank: 0
CPU name: Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
Frequency: 2.40 GHz
Logical CPU Count: 40
Operating System: Linux
Computer Name: p0218.1en.oss.edu

Re-finalize Sur...

INTEL ADVISOR 2018

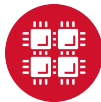


Profiling Python with cProfile

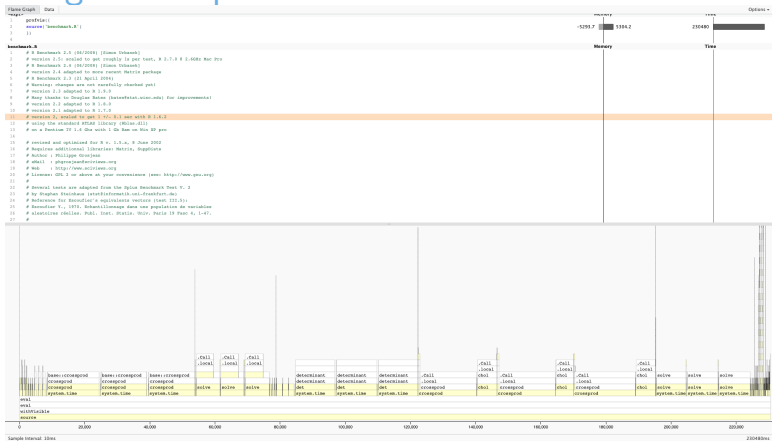
```
skhuvis@pitzer-login01:~$ python -m cProfile -s time poisson.py
320447 function calls (319459 primitive calls) in 8.904 seconds
```

Ordered by: internal time

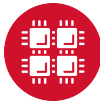
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	8.018	8.018	8.018	8.018	linalg.py:327(solve)
1	0.141	0.141	0.141	0.141	{matplotlib._deelaunay.linear_interpolate_grid}
1	0.093	0.093	0.096	0.096	numericatypes.py:81(<module>)
4	0.060	0.015	0.279	0.070	__init__.py:1(<module>)
1	0.038	0.038	8.313	8.313	poisson.py:37(poisson)
1	0.027	0.027	0.384	0.384	__init__.py:106(<module>)
1546	0.024	0.000	0.179	0.000	poisson.py:27(A_e)
3094	0.019	0.000	0.022	0.000	defmatrix.py:191(__getitem__)
1	0.018	0.018	0.036	0.036	overrides.py:1(<module>)
1546	0.017	0.000	0.030	0.000	linalg.py:486(inv)
1	0.013	0.013	0.013	0.013	hashlib.py:73(<module>)
23196	0.012	0.000	0.015	0.000	defmatrix.py:169(__array_finalize__)
1	0.011	0.011	0.016	0.016	__init__.py:15(<module>)
1	0.011	0.011	0.050	0.050	__init__.py:7(<module>)
1546	0.011	0.000	0.011	0.000	poisson.py:59(f)
1	0.011	0.011	0.023	0.023	numeric.py:1(<module>)
1	0.010	0.010	8.905	8.905	poisson.py:6(<module>)
4640	0.009	0.000	0.013	0.000	{numpy.concatenate}
15583	0.009	0.000	0.009	0.000	{numpy.array}
1547	0.009	0.000	0.029	0.000	index_tricks.py:36(ix_)
6184	0.009	0.000	0.009	0.000	{warnings.warn}
9278	0.009	0.000	0.017	0.000	shape_base.py:83(atleast_2d)
1546	0.009	0.000	0.017	0.000	linalg.py:2040(det)
1546	0.008	0.000	0.026	0.000	poisson.py:32(b_e)
1	0.007	0.007	0.019	0.019	npio.py:1(<module>)
1546	0.007	0.000	0.043	0.000	defmatrix.py:794(getI)
1	0.007	0.007	0.009	0.009	cbook.py:4(<module>)
2	0.007	0.003	0.020	0.010	__init__.py:45(<module>)
1546	0.007	0.000	0.007	0.000	{method 'reduce' of 'numpy.ufunc' objects}
1	0.006	0.006	0.025	0.025	index_tricks.py:1(<module>)



Profiling R with profvis



```
> install.packages('profvis')
> library('profvis')
> profvis({
  source('benchmark.R')
})
```



Resources to get your questions answered

FAQs: osc.edu/resources/getting_started/supercomputing_faqs

HOW TOs: osc.edu/resources/getting_started/howto

Performance Collection Guide:

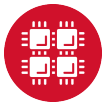
osc.edu/resources/getting_started/howto/howto_collect_performance_data_for_your_program

Office Hours:

go.osu.edu/rc-osc Tuesdays 1-3 p.m. or Wednesdays and Fridays
1-2:30 p.m. at Pomerene Hall

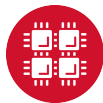
System updates:

- ▶ Read Message of the Day on login
- ▶ Follow [@HPCNotices](https://twitter.com/HPCNotices) on Twitter



Optimization and Performance Tuning Workshop

- ▶ October 29, 2019
- ▶ Present techniques for improving the performance of scientific software on High Performance Computing (HPC) systems such as those available at OSC.
- ▶ The focus will be on serial performance, including vectorization and cache utilization, with a brief mention of parallel computing.
- ▶ Topics covered:
 - ▶ Hardware overview
 - ▶ Important factors for good performance
 - ▶ Compiler optimizations
 - ▶ Profiling tools
- ▶ osc.edu/calendar/events/2019_10_29-optimization_performance_tuning_workshop
- ▶ Self-guided tutorial: https://www.osc.edu/resources/getting_started/howto/howto_tune_performance



OH·TECH

Ohio Technology Consortium
A Division of the Ohio Department of Higher Education


 info@osc.edu

 twitter.com/osc

 facebook.com/ohiosupercomputercenter

 osc.edu

 oh-tech.org/blog

 linkedin.com/company/ohio-supercomputer-center

