

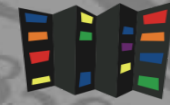
# Debugging, benchmarking, tuning i.e. software development tools

*Martin Čuma*

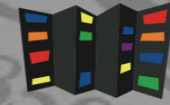
*Center for High Performance Computing*

*University of Utah*

*m.cuma@utah.edu*

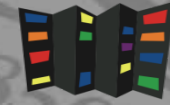


- Development environments
- Compilers
- Version control
- Debuggers
- Profilers
- Runtime monitoring
- Benchmarking

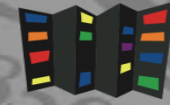


# PROGRAMMING TOOLS

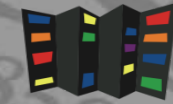




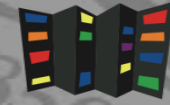
- Open source
  - GNU
  - Open64, clang
- Commercial
  - Intel
  - Portland Group (PGI, owned by Nvidia)
  - Vendors (IBM XL, Cray)
  - Others (Absoft, CAPS, PathScale)



- Languages
  - C/C++ - GNU, Intel, PGI
  - Fortran – GNU, Intel, PGI
- Interpreters
  - Matlab – has its own ecosystem
  - Java – reasonable ecosystem, not so popular in HPC, popular in HTC
  - Python – attempts to have its own ecosystem, some tools can plug into Python (e.g. Intel VTune)



- Language extensions
  - OpenMP (4.0+\*) – GNU, Intel\*, PGI
  - OpenACC – PGI, GNU very experimental
  - CUDA – Nvidia GCC, PGI Fortran
- Libraries
  - Intel Math Kernel Library (MKL)
  - PGI packages open source (OpenBLAS?).

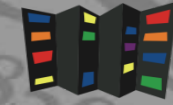


- Copies of programs
  - Good enough for simple code and quick tests/changes
- Version control software
  - Allow code merging, branching, etc
  - Essential for collaborative development
  - RCS, CVS, SVN
  - Git – integrated web services, free for open source, can run own server for private code

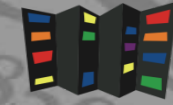




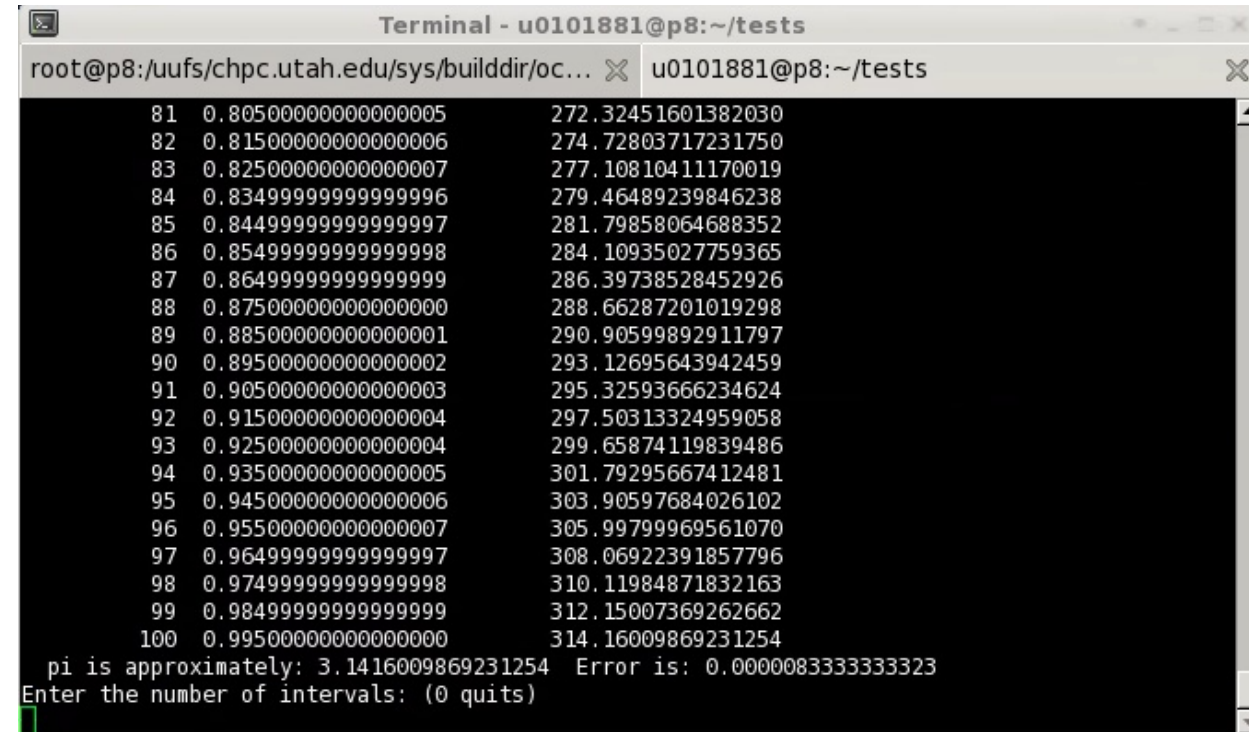
# DEBUGGING



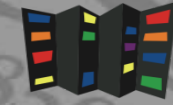
- Crashes
  - Segmentation faults (bad memory access)
    - often writes core file – snapshot of memory at the time of the crash
  - Wrong I/O (missing files)
  - Hardware failures
- Incorrect results
  - Reasonable but incorrect results
  - NaNs – not a numbers – division by 0, ...



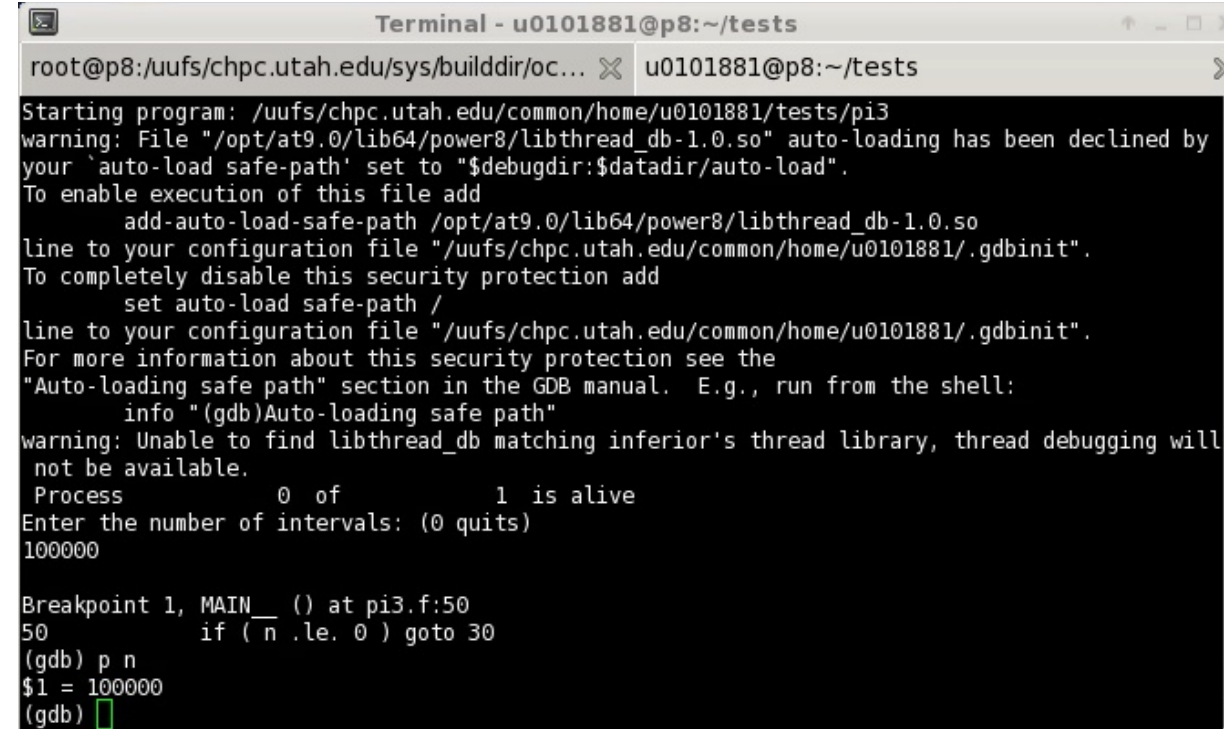
- Write variables of interest into the stdout or file
- Simplest but cumbersome
  - Need to recompile and rerun
  - Need to browse through potentially large output



```
Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/buildir/oc... x u0101881@p8:~/tests
81 0.80500000000000005      272.32451601382030
82 0.81500000000000006      274.72803717231750
83 0.82500000000000007      277.10810411170019
84 0.83499999999999996      279.46489239846238
85 0.84499999999999997      281.79858064688352
86 0.85499999999999998      284.10935027759365
87 0.86499999999999999      286.39738528452926
88 0.87500000000000000      288.66287201019298
89 0.88500000000000001      290.90599892911797
90 0.89500000000000002      293.12695643942459
91 0.90500000000000003      295.32593666234624
92 0.91500000000000004      297.50313324959058
93 0.92500000000000004      299.65874119839486
94 0.93500000000000005      301.79295667412481
95 0.94500000000000006      303.90597684026102
96 0.95500000000000007      305.99799969561070
97 0.96499999999999997      308.06922391857796
98 0.97499999999999998      310.11984871832163
99 0.98499999999999999      312.15007369262662
100 0.99500000000000000     314.16009869231254
pi is approximately: 3.1416009869231254 Error is: 0.0000083333333323
Enter the number of intervals: (0 quits)
```

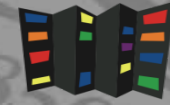


- Text only, e.g. gdb, idb
- Need to remember commands or their abbreviations
- Need to know lines in the code (or have it opened in other window)
- Useful for quick code checking on compute nodes and core dump analysis

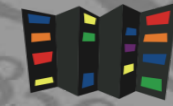


```
Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/buildir/oc... x u0101881@p8:~/tests
Starting program: /uufs/chpc.utah.edu/common/home/u0101881/tests/pi3
warning: File "/opt/at9.0/lib64/power8/libthread_db-1.0.so" auto-loading has been declined by
your 'auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /opt/at9.0/lib64/power8/libthread_db-1.0.so
line to your configuration file "/uufs/chpc.utah.edu/common/home/u0101881/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/uufs/chpc.utah.edu/common/home/u0101881/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
  info "(gdb)Auto-loading safe path"
warning: Unable to find libthread_db matching inferior's thread library, thread debugging will
not be available.
Process          0 of          1 is alive
Enter the number of intervals: (0 quits)
100000

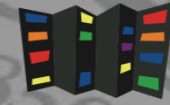
Breakpoint 1, MAIN__ () at pi3.f:50
50      if ( n .le. 0 ) goto 30
(gdb) p n
$1 = 100000
(gdb) █
```



- Have graphical user interface
- Some free, mostly commercial
- Eclipse CDT (C/C++ Development Tooling), PTP (Parallel Tools Platform) - free
- PGI's pdbg – part of PGI compiler suite
- Intel development tools
- Rogue Wave Totalview - commercial
- Allinea DDT - commercial



- The only real alternative for parallel or accelerator debugging
- Cost a lot of money (thousands of \$), but, worth it
- We have Totalview license (for historical reasons), 32 tokens enough for our needs (renewal ~\$1500/yr).
- XSEDE systems have DDT.



## 1. Compile binary with debugging information

- flag -g

```
gcc -g test.f -o test
```

## 2. Load module and run Totalview

```
module load totalview
```

- TV + executable

```
totalview executable
```

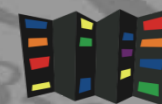
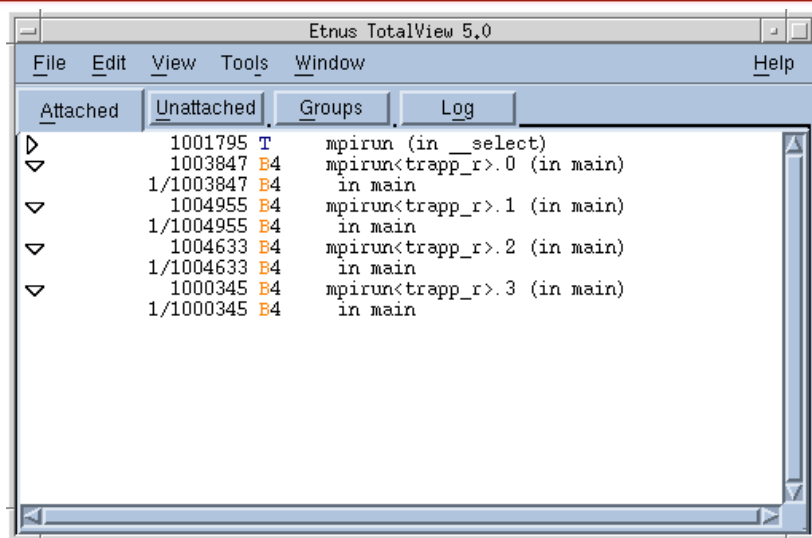
- TV + core file

```
totalview executable core_file
```

- Run TV and choose what to debug in a startup dialog

```
totalview
```



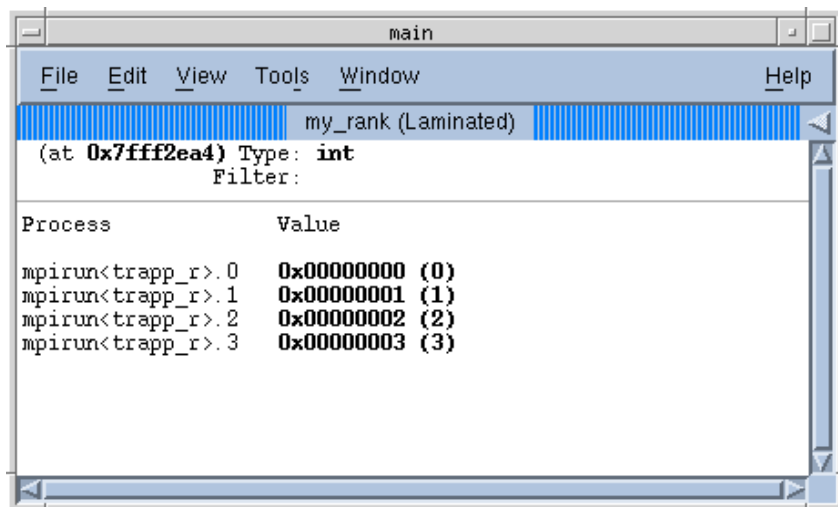



Etnus TotalView 5.0

File Edit View Tools Window Help

Attached Unattached Groups Log

- 1001795 T mpirun (in \_\_select)
- 1003847 B4 mpirun<trapp\_r>.0 (in main)
- 1/1003847 B4 in main
- 1004955 B4 mpirun<trapp\_r>.1 (in main)
- 1/1004955 B4 in main
- 1004633 B4 mpirun<trapp\_r>.2 (in main)
- 1/1004633 B4 in main
- 1000345 B4 mpirun<trapp\_r>.3 (in main)
- 1/1000345 B4 in main



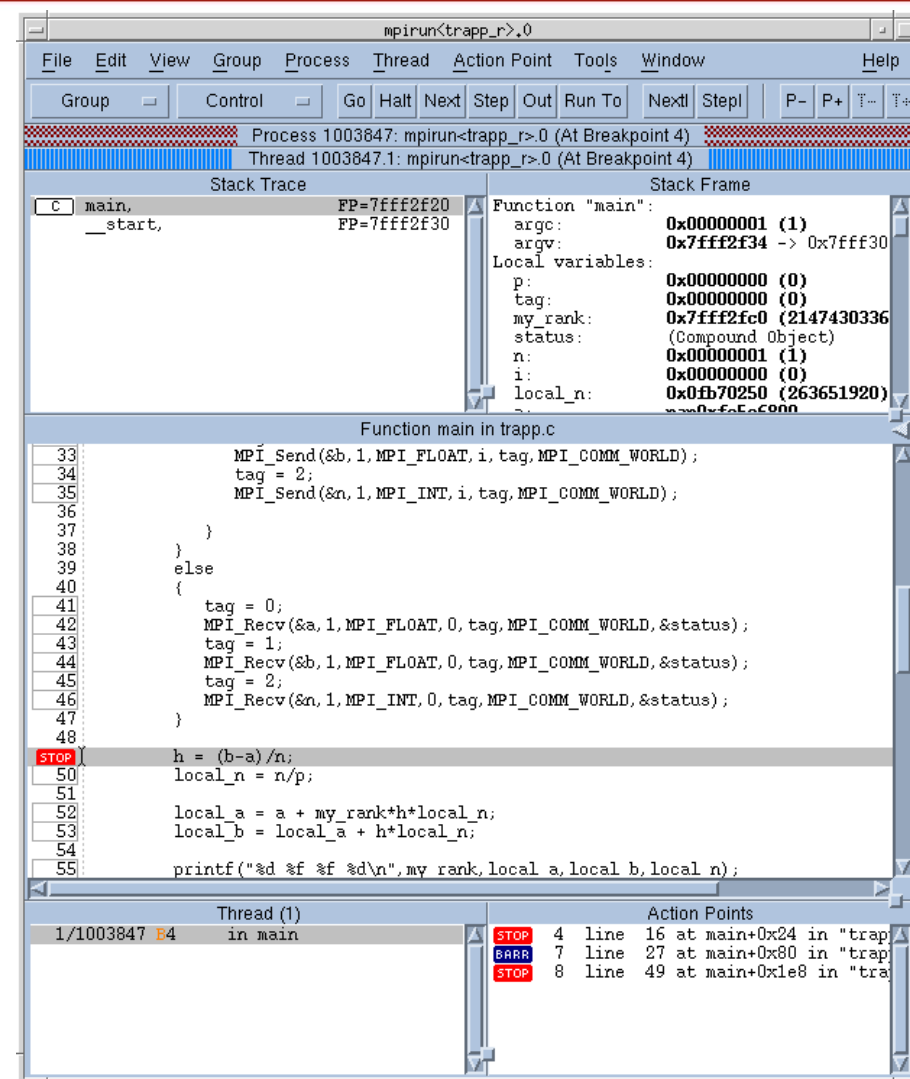
main

File Edit View Tools Window Help

my\_rank (Laminated)

(at 0x7fff2ea4) Type: int  
Filter:

Process	Value
mpirun<trapp_r>.0	0x00000000 (0)
mpirun<trapp_r>.1	0x00000001 (1)
mpirun<trapp_r>.2	0x00000002 (2)
mpirun<trapp_r>.3	0x00000003 (3)



mpirun<trapp\_r>.0

File Edit View Group Process Thread Action Point Tools Window Help

Group Control Go Halt Next Step Out Run To Next| Stepl P- P+ T- T+

Process 1003847: mpirun<trapp\_r>.0 (At Breakpoint 4)  
Thread 1003847.1: mpirun<trapp\_r>.0 (At Breakpoint 4)

Stack Trace

FP	Function
7fff2f20	main, _start
7fff2f30	

Stack Frame

Function "main":

- argc: 0x00000001 (1)
- argv: 0x7fff2f34 -> 0x7fff30
- Local variables:
- p: 0x00000000 (0)
- tag: 0x00000000 (0)
- my\_rank: 0x7fff2fc0 (2147430336)
- status: (Compound Object)
- n: 0x00000001 (1)
- i: 0x00000000 (0)
- local\_n: 0x0fb70250 (263651920)

Function main in trapp.c

```

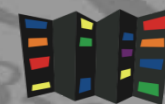
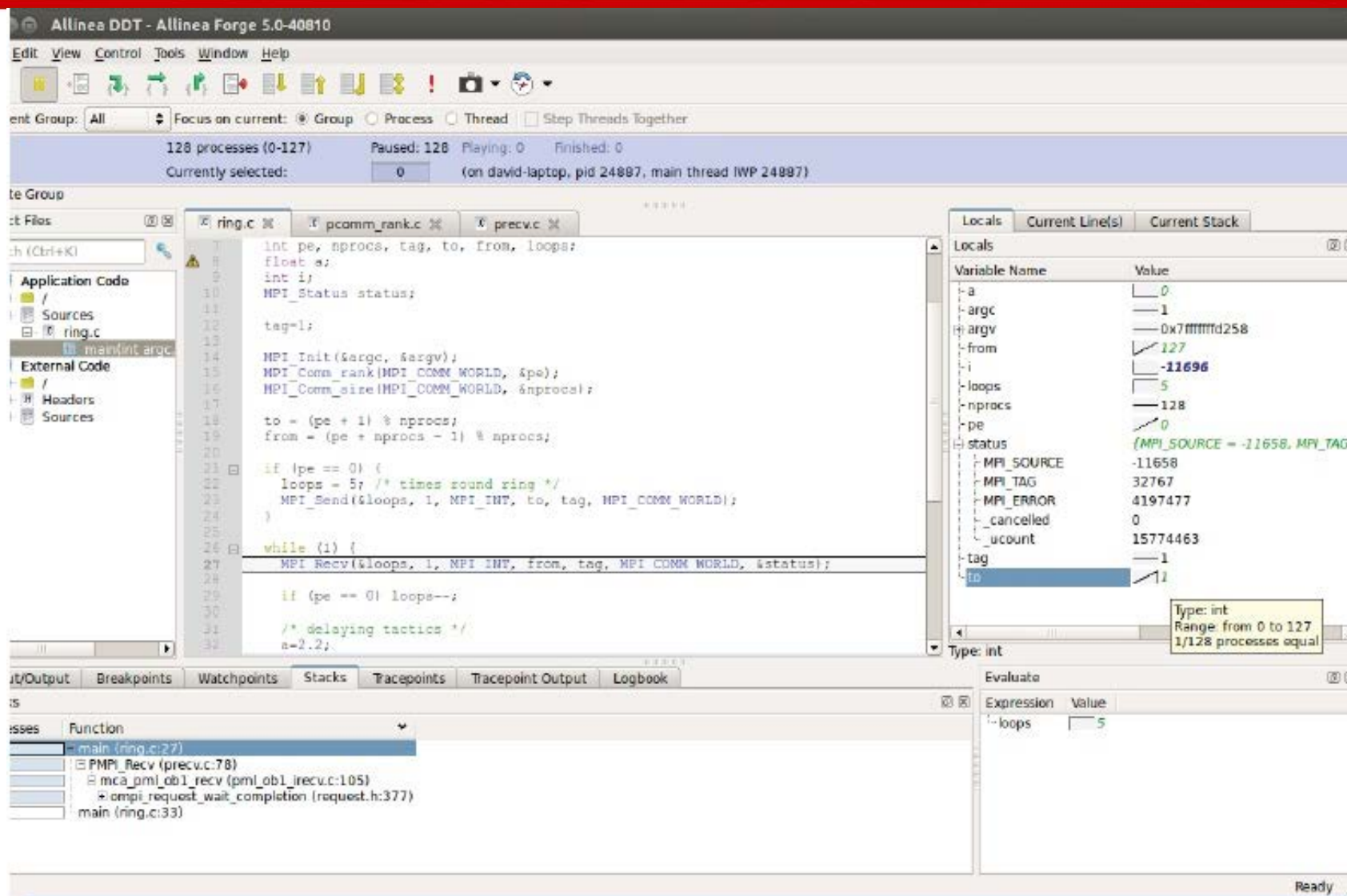
33 MPI_Send(&b, 1, MPI_FLOAT, i, tag, MPI_COMM_WORLD);
34 tag = 2;
35 MPI_Send(&n, 1, MPI_INT, i, tag, MPI_COMM_WORLD);
36
37 }
38 }
39 else
40 {
41 tag = 0;
42 MPI_Recv(&a, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &status);
43 tag = 1;
44 MPI_Recv(&b, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &status);
45 tag = 2;
46 MPI_Recv(&n, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
47
48
49 h = (b-a)/n;
50 local_n = n/p;
51
52 local_a = a + my_rank*h*local_n;
53 local_b = local_a + h*local_n;
54
55 printf("%d %f %f %d\n", my rank, local a, local b, local n);

```

Thread (1)

Thread	Action Points
1/1003847 B4 in main	STOP 4 line 16 at main+0x24 in "trap
	BARR 7 line 27 at main+0x80 in "trap
	STOP 8 line 49 at main+0x1e8 in "tra



Allinea DDT - Allinea Forge 5.0-40810

Edit View Control Tools Window Help

ent Group: All Focus on current: Group Process Thread Step Threads Together

128 processes (0-127) Paused: 128 Playing: 0 Finished: 0  
Currently selected: 0 (on david-laptop, pid 24887, main thread IWP 24887)

File: ring.c

```

1  int pe, nprocs, tag, to, from, loops;
2  float a;
3  int i;
4  MPI_Status status;
5
6  tag=1;
7
8  MPI_Init(&argc, &argv);
9  MPI_Comm_rank(MPI_COMM_WORLD, &pe);
10 MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
11
12 to = (pe + 1) % nprocs;
13 from = (pe + nprocs - 1) % nprocs;
14
15 if (pe == 0) {
16     loops = 5; /* times round ring */
17     MPI_Send(&loops, 1, MPI_INT, to, tag, MPI_COMM_WORLD);
18 }
19
20 while (1) {
21     MPI_Recv(&loops, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
22
23     if (pe == 0) loops--;
24
25     /* delaying tactics */
26     a=2.2;
27 }

```

Locals

Variable Name	Value
a	2.2
argc	1
argv	0x7ffffffd258
from	127
i	-11696
loops	5
nprocs	128
pe	0
status	{MPI_SOURCE = -11658, MPI_TAG = 32767, MPI_ERROR = 4197477, _cancelled = 0, _ucount = 15774463}
tag	1
to	1

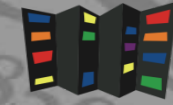
Call Stack

Function
main (ring.c:27)
PMPI_Recv (precv.c:78)
mca_pml_ob1_recv (pml_ob1_recv.c:105)
ompi_request_wait_completion (request.h:377)
main (ring.c:33)

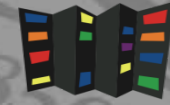
Expression Value

Expression	Value
loops	5

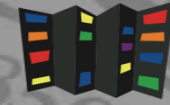
Ready



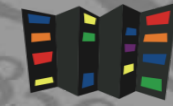
- Data examination
  - view data in the variable windows
  - change the values of variables
  - modify display of the variables
  - visualize data
- Action points
  - breakpoints and barriers (static or conditional)
  - watchpoints
  - evaluation of expressions



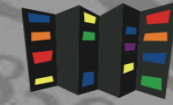
- Automatic attachment of child processes
- Create process groups
- Share breakpoints among processes
- Process barrier breakpoints
- Process group single-stepping
- View variables across procs/threads
- Display MPI message queue state



- Memoryscape
  - Dynamic memory debugging tool
- Replay Engine
  - Allows to reversely debug the code
- Accelerator debugging
  - CUDA and OpenACC

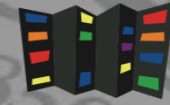


- Compilers check for syntax errors
  - lint based tools
  - Runtime checks through compiler flags (-fbounds-check, -check\*, -Mbounds)
- DDT has a built in syntax checker
  - Matlab does too
- Memory checking tools - many errors are due to bad memory management
  - valgrind – easy to use, many false positives
  - Intel Inspector – intuitive GUI



- We have a 2 concurrent user license
  - One license locks all the tools
  - Cost ~\$2000/year
- Tools for all stages of development
  - Compilers and libraries
  - Verification tools
  - Profilers
- More info

<https://software.intel.com/en-us/intel-parallel-studio-xe>



- Thread checking
  - Data races and deadlocks
- Memory checker
  - Like leaks or corruption
  - Good alternative to Totalview MemoryScape
- Standalone or GUI integration
- More info

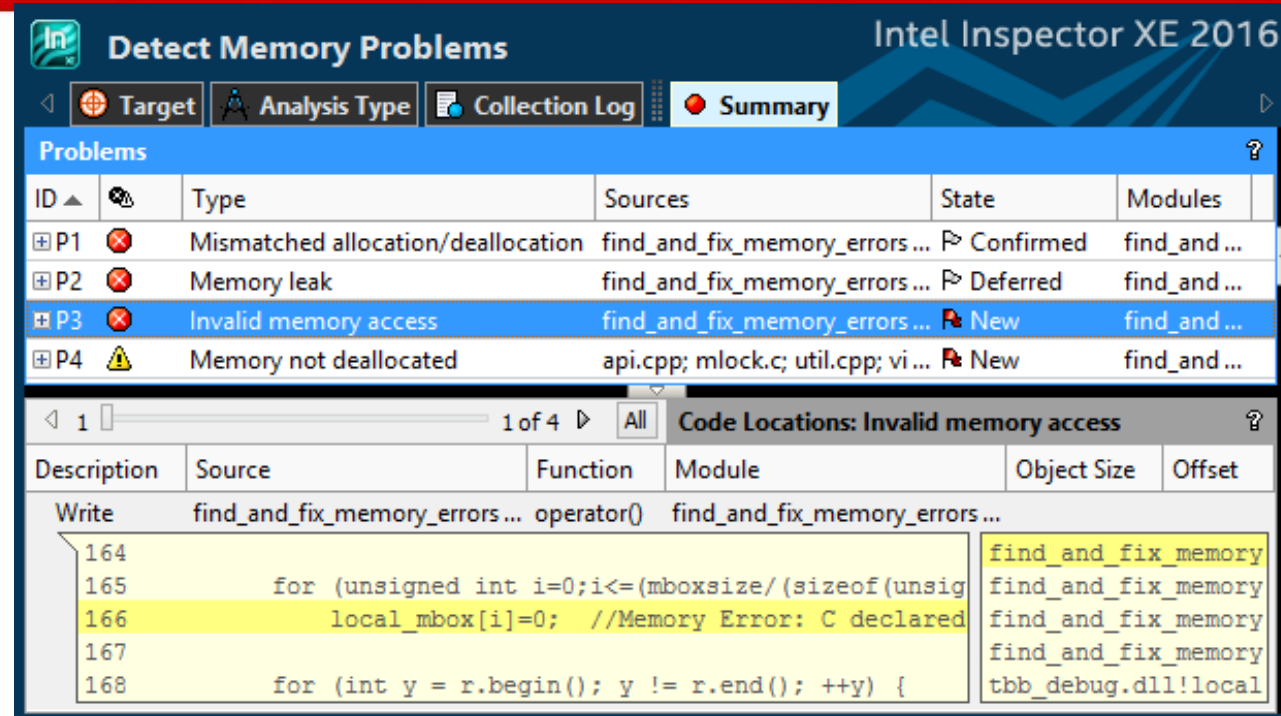
<http://software.intel.com/en-us/intel-inspector-xe/>





- Source the environment  
module load inspectorxe
- Compile with `-tcheck -g`  
`ifort -openmp -tcheck -g trap.f`
- Run `tcheck`  
`inspxe-gui` – graphical user interface  
`inspxe-cl` – command line
- Tutorial

<https://software.intel.com/en-us/articles/inspectorxe-tutorials>



**Detect Memory Problems** Intel Inspector XE 2016

Target Analysis Type Collection Log Summary

ID	Type	Sources	State	Modules
P1	Mismatched allocation/deallocation	find_and_fix_memory_errors...	Confirmed	find_and...
P2	Memory leak	find_and_fix_memory_errors...	Deferred	find_and...
P3	Invalid memory access	find_and_fix_memory_errors...	New	find_and...
P4	Memory not deallocated	api.cpp; mlock.c; util.cpp; vi...	New	find_and...

Code Locations: Invalid memory access

Description	Source	Function	Module	Object Size	Offset
Write	find_and_fix_memory_errors...	operator()	find_and_fix_memory_errors...		
	164				find_and_fix_memory
	165	for (unsigned int i=0;i<=(mboxsize/(sizeof(unsig			find_and_fix_memory
	166	local_mbox[i]=0; //Memory Error: C declared			find_and_fix_memory
	167				find_and_fix_memory
	168	for (int y = r.begin(); y != r.end(); ++y) {			tbb_debug.dll!local



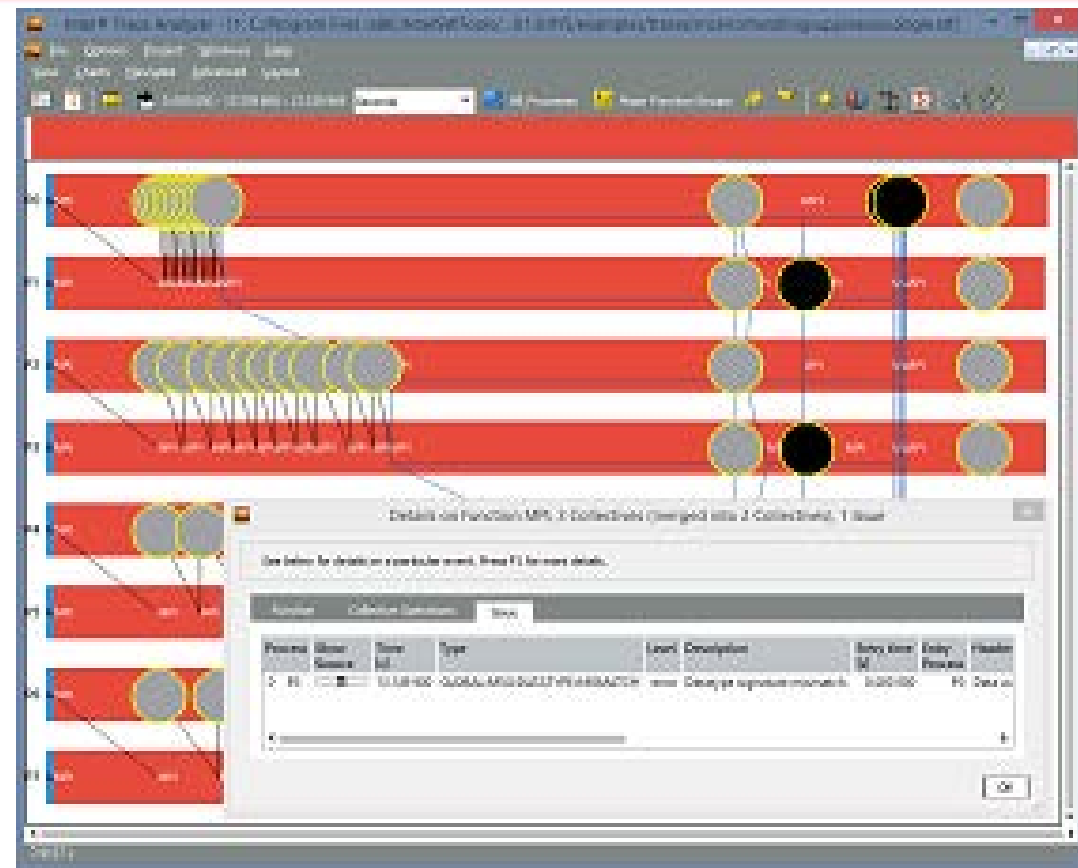


- MPI profiler and correctness checker
- Detects violations of MPI standard and errors in execution environment
- To use correctness checker

```
module load intel impi itac
setenv VT_CHECK_TRACING 0
mpirun -check-mpi -n 4 ./myApp
```

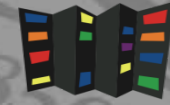
- ITAC documentation

<https://software.intel.com/en-us/intel-trace-analyzer-support/documentation>

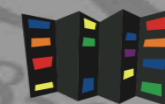




# PROFILING



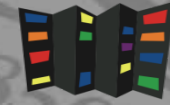
- Evaluate performance
- Find the performance bottlenecks
  - Inefficient programming
  - Memory or I/O bottlenecks
  - Parallel scaling



- Time program runtime
  - get an idea on time to run and parallel scaling
- Many programs include benchmark problems
  - Some also accessible via “make test”
- Consider scripts, especially if doing parallel performance evaluation

```
Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/buildidir/oc... x u0101881@p8:~/tests

[u0101881@p8 ~/tests]$ mpicc -O3 cpi.c -o cpi
[u0101881@p8 ~/tests]$ time mpirun -np 16 ./cpi
pi is approximately 3.1415926535898451, Error is 0.00000000000000520
wall clock time = 2.338931
Process 0 before finalize
37.959u 0.194s 0:02.40 1589.1% 0+0k 0+0io 0pf+0w
[u0101881@p8 ~/tests]$ time mpirun -bind-to numa -map-by numa -np 16 ./cpi
pi is approximately 3.1415926535898451, Error is 0.00000000000000520
wall clock time = 2.296053
Process 0 before finalize
36.853u 0.371s 0:02.52 1476.9% 0+0k 0+0io 0pf+0w
[u0101881@p8 ~/tests]$ time mpirun -bind-to core -map-by core -np 16 ./cpi
pi is approximately 3.1415926535898451, Error is 0.00000000000000520
wall clock time = 0.513617
Process 0 before finalize
8.268u 0.249s 0:00.72 1180.5% 0+0k 0+0io 0pf+0w
[u0101881@p8 ~/tests]$ time mpirun -bind-to numa -map-by numa -np 1 ./cpi
pi is approximately 3.1415926535899708, Error is 0.0000000000001776
wall clock time = 4.488950
Process 0 before finalize
4.553u 0.133s 0:04.69 99.7% 0+0k 0+0io 0pf+0w
[u0101881@p8 ~/tests]$
```



- Hardware counters
  - count events from CPU perspective (# of flops, memory loads, etc)
  - usually need Linux kernel module installed (>2.6.31 has it)
- Statistical profilers (sampling)
  - interrupt program at given intervals to find what routine/line the program is in
- Event based profilers (tracing)
  - collect information on each function call

- CPUs include counters to count important events
  - Flops, instructions, cache/memory access
  - Access through kernel or PAPI (Performance Application Programming Interface)
- Tools to analyze the counters
  - perf - hardware counter collection, part of Linux
  - oprofile – profiler + hw counters
  - Intel VTune
- Drawback – harder to analyze the profiling results (exc. VTune)

```
Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/buildir/oc... x u0101881@p8:~/tests
[u0101881@p8 ~/tests]$ mpicc -O3 cpi.c -o cpi
[u0101881@p8 ~/tests]$ perf stat mpirun -bind-to core -map-by core -np 1 ./cpi
pi is approximately 3.1415926535899708, Error is 0.0000000000001776
wall clock time = 4.976919
Process 0 before finalize

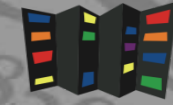
Performance counter stats for 'mpirun -bind-to core -map-by core -np 1 ./cpi':

   5176.211056      task-clock (msec)      #    0.999 CPUs utilized
                   78      context-switches      #    0.015 K/sec
                    3      cpu-migrations        #    0.001 K/sec
                  4,255      page-faults          #    0.822 K/sec
  17,641,344,721      cycles                 #    3.408 GHz          (66.76%)
   16,802,571      stalled-cycles-frontend #    0.10% frontend cycles idle (50.17%)
  12,934,161,831      stalled-cycles-backend #    73.32% backend cycles idle (50.16%)
  13,393,245,960      instructions           #    0.76 insns per cycle
                   #    0.97 stalled cycles per insn (66.78%)
   1,083,269,385      branches               # 209.278 M/sec        (49.84%)
   1,479,078      branch-misses          #    0.14% of all branches (50.10%)

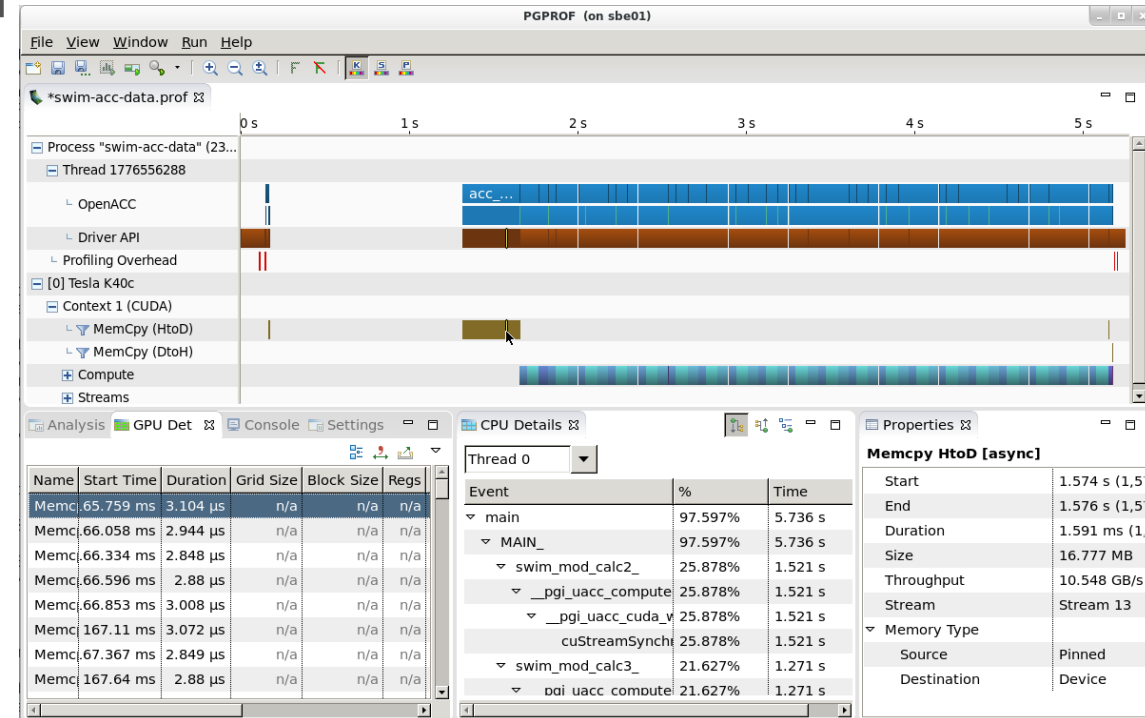
   5.180999586 seconds time elapsed

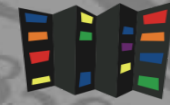
[u0101881@p8 ~/tests]$
```





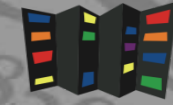
- Discover inefficient programming
- Computer architecture slowdowns
- Compiler optimizations evaluation
- gprof
- Compiler vendor supplied (e.g. pgprof, nvvp)
- Intel tools on serial programs
  - AdvisorXE, VTune



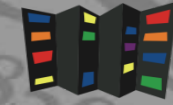


- HPC Toolkit
  - A few years old, did not find it as straightforward to use
- TAU (Tuning and Analysis Utilities)
  - Lots of features, which makes the learning curve slow
- Scalasca
  - Developed by European consortium, did not try yet





- Intel Parallel Studio XE 2016 Cluster Edition
  - Compilers (C/C++, Fortran)
  - Math library (MKL)
  - Threading library (TBB)
  - Thread design and prototype (Advisor)
  - Memory and thread debugging (Inspector)
  - Profiler (VTune Amplifier)
  - MPI library (Intel MPI)
  - MPI analyzer and profiler (ITAC)

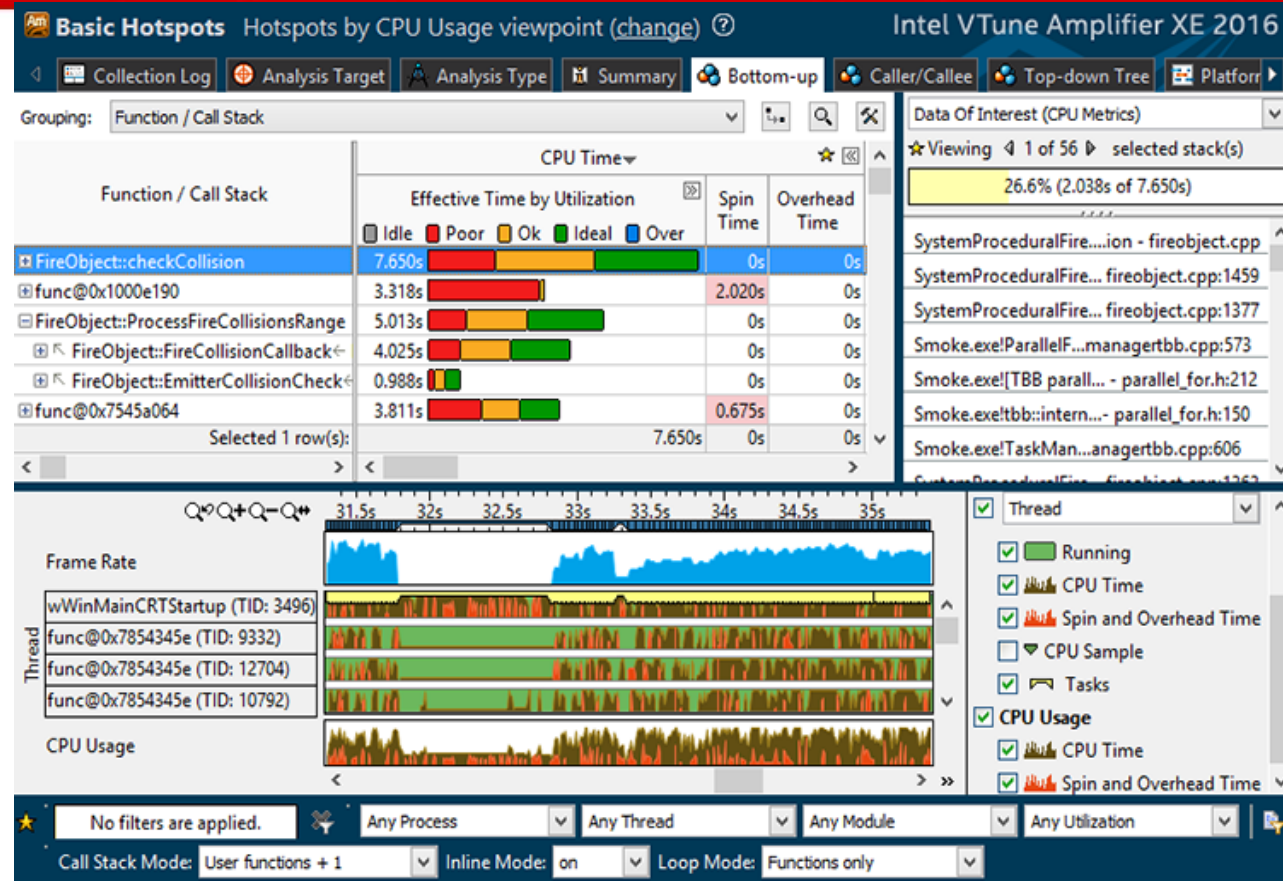


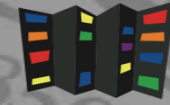
- Serial and parallel profiler
  - Multicore support for OpenMP and OpenCL on CPUs, GPUs and Xeon Phi
- Quick identification of performance bottlenecks
  - Various analyses and points of view in the GUI
  - Makes choice of analysis and results inspection easier
- GUI and command line use
- More info

<https://software.intel.com/en-us/intel-vtune-amplifier-xe>

- Source the environment  
module load vtune
- Run VTune  
amplxe-gui – GUI  
amplxe-cl – CLI
- Can be used also for remote profiling (e.g. on Xeon Phi)
- Tuning guides for specific architectures

<https://software.intel.com/en-us/articles/processor-specific-performance-analysis-papers>





- Vectorization advisor
  - Identify loops that benefit from vectorization, what is blocking efficient vectorization and explore benefit of data reorganization
- Thread design and prototyping
  - Analyze, design, tune and check threading design without disrupting normal development
- More info

<http://software.intel.com/en-us/intel-advisor-xe/>



- Source the environment

module load advisorxe

- Run Advisor

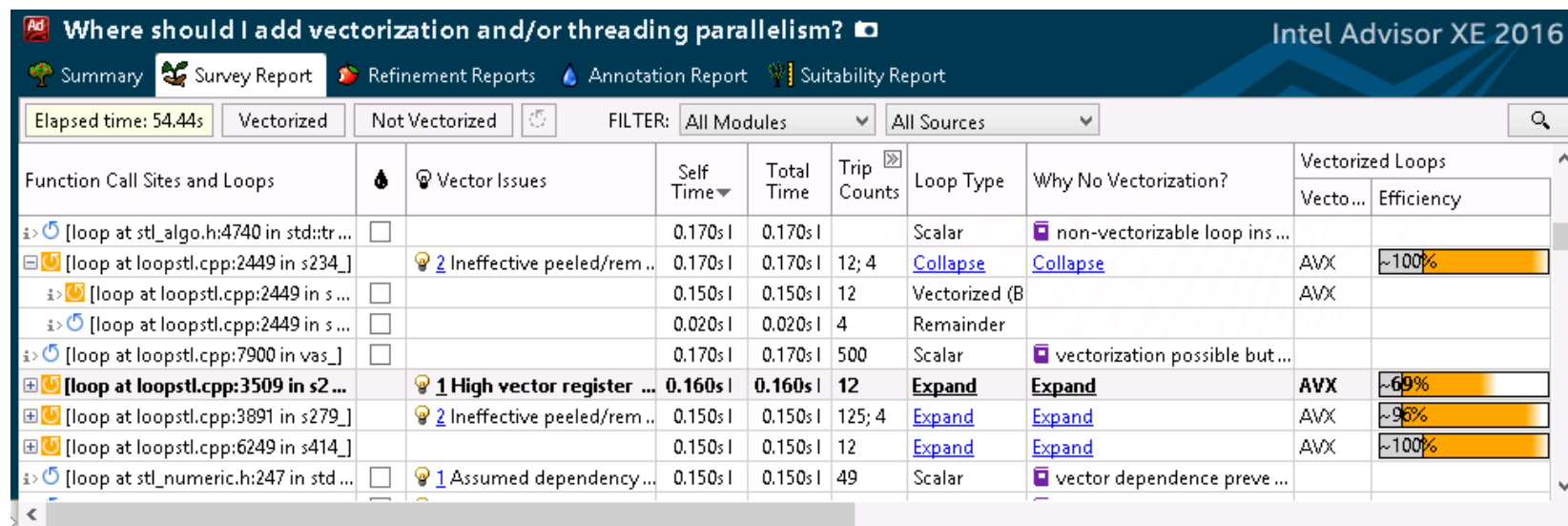
advixe-gui – GUI

advixe-cl – CLI

- Create project and choose appropriate modeling

- Getting started guide

<https://software.intel.com/en-us/get-started-with-advisor>

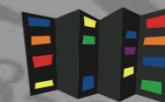


Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules All Sources

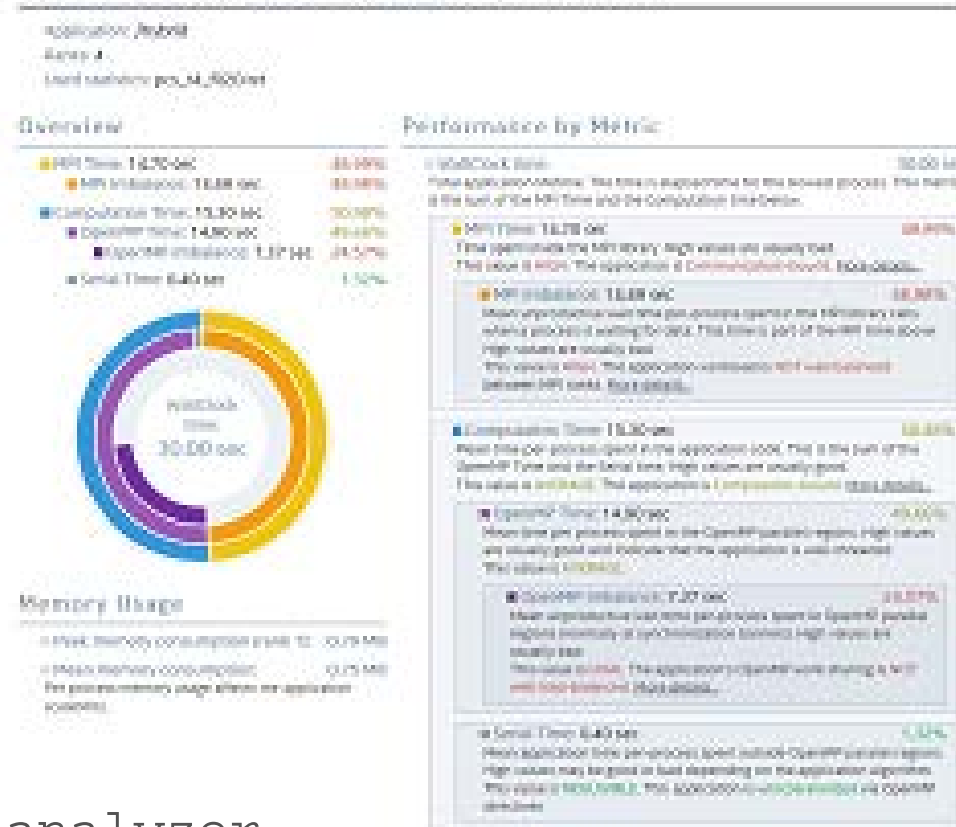
Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops	
							Vecto...	Efficiency
[loop at stl_algo.h:4740 in std:tr...]		0.170s	0.170s		Scalar	non-vectorizable loop ins ...		
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%
[loop at loopstl.cpp:2449 in s...]		0.150s	0.150s	12	Vectorized (B		AVX	
[loop at loopstl.cpp:2449 in s...]		0.020s	0.020s	4	Remainder			
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...		
[loop at loopstl.cpp:3509 in s2...]	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX	~69%
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%
[loop at stl_numeric.h:247 in std...]	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve...		



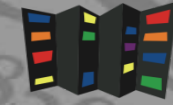
- MPI profiler
  - traces MPI code
  - identifies communication inefficiencies
- Collector collects the data and Analyzer visualizes them
- More info

<https://software.intel.com/en-us/intel-trace-analyzer>

## MPI Performance Snapshot Summary







- Source the environment

```
module load itac
```

- Using Intel compilers, can compile with `-trace`

```
mpifort -openmp -trace trap.f
```

- Run MPI code

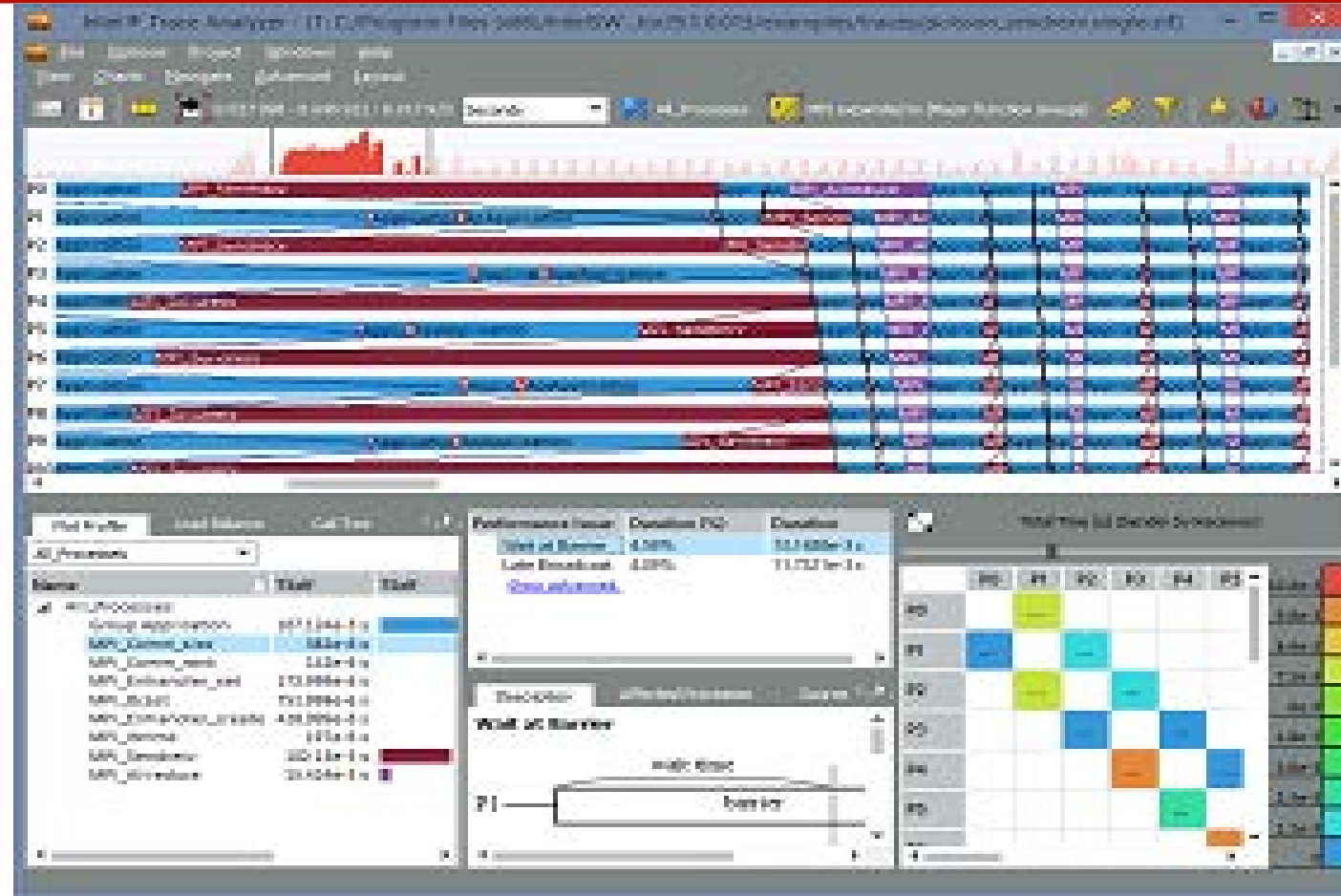
```
mpirun -trace -n 4 ./a.out
```

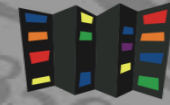
- Run visualizer

```
traceanalyzer a.out.stf &
```

- Getting started guide

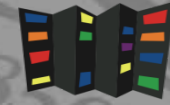
<https://software.intel.com/en-us/get-started-with-itac-for-linux>



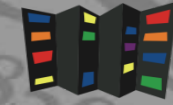


# RUNTIME MONITORING





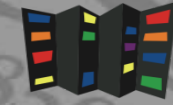
- Make sure program is running right
  - Hardware problems
  - Correct parallel mapping / process affinity
- Careful about overhead



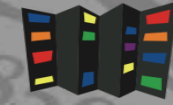
- Self checking
  - ssh to node(s), run “top”, or look at “sar” logs
  - SLURM (or other scheduler) logs and statistics
- Tools
  - XDMoD – XSEDE Metrics on Demand (through SUPReMM module)
  - REMORA - REsource MOnitoring for Remote Applications



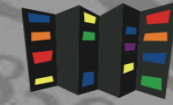
# BENCHMARKING



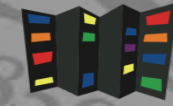
- Evaluate system's performance
  - Testing new hardware
- Verify correct hardware and software installation
  - New cluster/node deployment
    - There are tools for cluster checking (Intel Cluster Checker, cluster distros, ...)
  - Checking newly built programs
    - Sometimes we leave this to the users



- Simple synthetic benchmarks
  - FLOPS, STREAM
- Synthetic benchmarks
  - HPL – High Performance Linpack – dense linear algebra problems – cache friendly
  - HPCC – HPC Challenge Benchmark – collection of dense, sparse and other (FFT) benchmarks
  - NPB – NAS Parallel Benchmarks – mesh based solvers – OpenMP, MPI, OpenACC implementations



- Real applications benchmarks
  - Depend on local usage
  - Gaussian, VASP
  - Amber, LAMMPS, NAMD, Gromacs
  - ANSYS, Abaqus, StarCCM+
  - Own codes
- Script if possible
  - A lot of combinations of test cases vs. number of MPI tasks/OpenMP cores

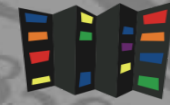


- Whole cluster
  - Some vendors have cluster verification tools
  - We have a set of scripts that run basic checks and HPL at the end
- New cluster nodes
  - Verify received hardware configuration, then rack
  - Basic system tests (node health check)
  - HPL – get expected performance per node (CPU or memory issues), or across more nodes (network issues)





# BACKUP



- Totalview
- Advisor
- Inspector
- VTune