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

Martin Čuma
Center for High Performance Computing
University of Utah
m.cuma@utah.edu

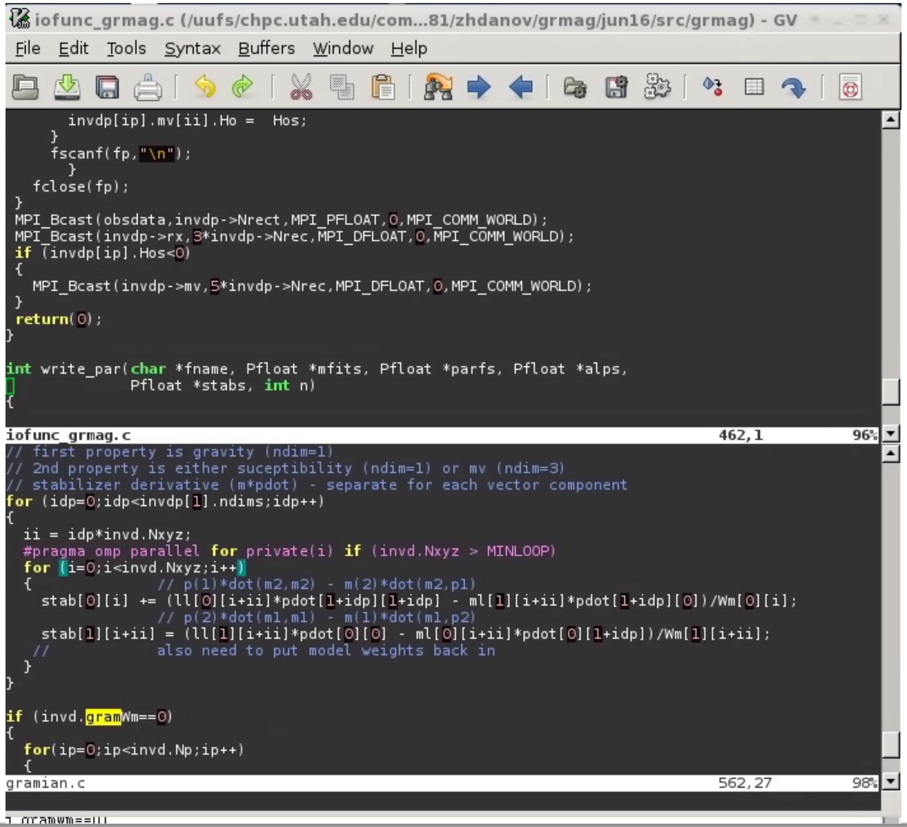
SW development tools

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

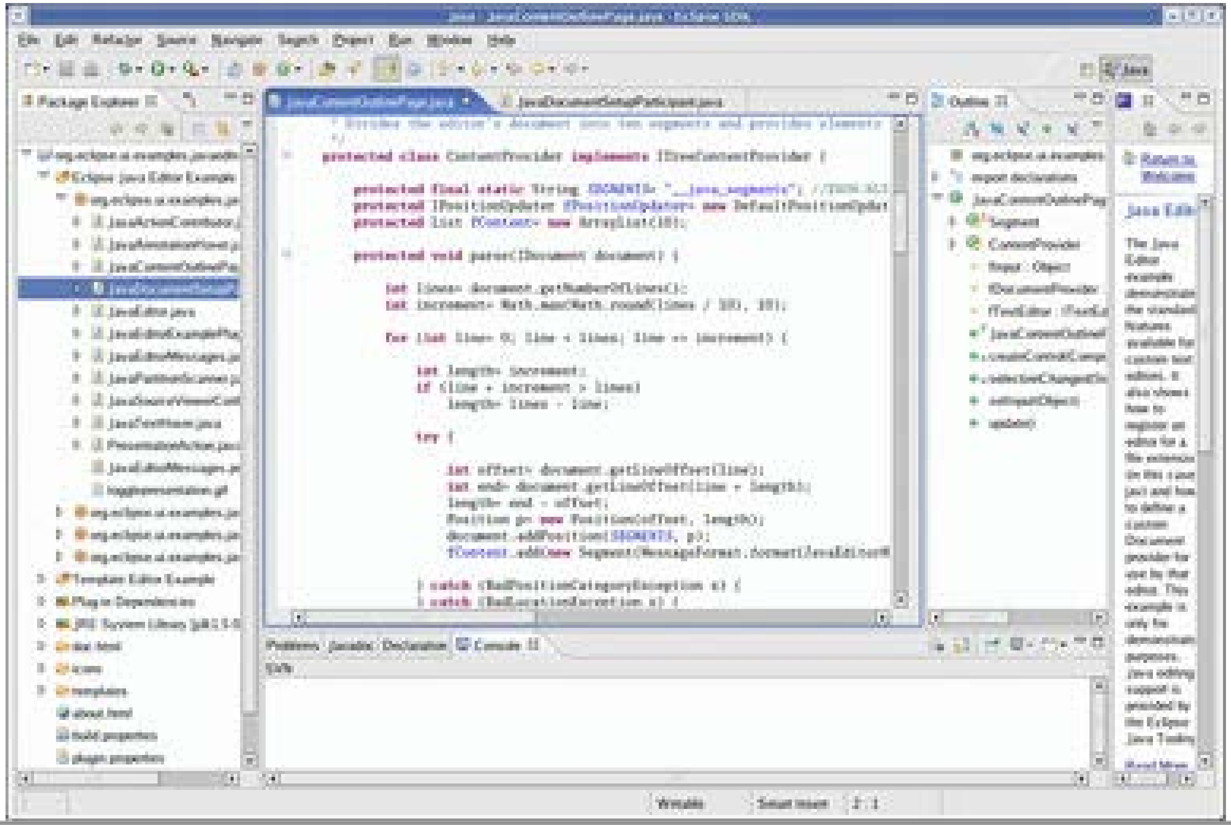
PROGRAMMING TOOLS

Program editing

- Text editors
  - vim, emacs, atom
- IDEs
  - Visual *, Eclipse


iofunc_grmag.c (/uufs/chpc.utah.edu/com...81/zhdanov/grmag/jun16/src/grmag) - GV
File Edit Tools Syntax Buffers Window Help

```
      invdp[ip].mv[ii].Ho =  Hos;
     }
     fscanf(fp,"\n");
       }
   fclose(fp);
 }
 MPI_Bcast(obsdata,invdp->Nrect,MPI_PFLOAT,0,MPI_COMM_WORLD);
 MPI_Bcast(invdp->rx,3*invdp->Nrec,MPI_DFLOAT,0,MPI_COMM_WORLD);
 if (invdp[ip].Hos<0)
 {
   MPI_Bcast(invdp->mv,5*invdp->Nrec,MPI_DFLOAT,0,MPI_COMM_WORLD);
 }
 return(0);
}

int write_par(char *fname, Pfloat *mfits, Pfloat *parfs, Pfloat *alps,
              Pfloat *stabs, int n)
{
iofunc_grmag.c                                                  462,1          96%
// first property is gravity (ndim=1)
// 2nd property is either suceptibility (ndim=1) or mv (ndim=3)
// stabilizer derivative (m*pdot) - separate for each vector component
for (idp=0;idp<invdp[1].ndims;idp++)
{
  ii = idp*invd.Nxyz;
  #pragma omp parallel for private(i) if (invd.Nxyz > MINLOOP)
  for (i=0;i<invd.Nxyz;i++)
  {               // p(1)*dot(m2,m2) - m(2)*dot(m2,p1)
    stab[0][i] += (ll[0][i+ii]*pdot[1+idp][1+idp] - ml[1][i+ii]*pdot[1+idp][0])/Wm[0][i];
                  // p(2)*dot(m1,m1) - m(1)*dot(m1,p2)
    stab[1][i+ii] = (ll[1][i+ii]*pdot[0][0] - ml[0][i+ii]*pdot[0][1+idp])/Wm[1][i+ii];
   //             also need to put model weights back in
  }
}

if (invd.gramWm==0)
{
  for(ip=0;ip<invd.Np;ip++)
  {
gramian.c                                                       562,27         98%
```



```
 * Divides the editor's document into ten segments and provides elements
 */
protected class ContentProvider implements ITreeContentProvider {

    protected final static String SEGMENTS= "__java_segments"; //$NON-NLS-1$
    protected IPositionUpdater fPositionUpdater= new DefaultPositionUpdat
    protected List fContent= new ArrayList(10);

    protected void parse(IDocument document) {

        int lines= document.getNumberOfLines();
        int increment= Math.max(Math.round(lines / 10), 10);

        for (int line= 0; line < lines; line += increment) {

            int length= increment;
            if (line + increment > lines)
                length= lines - line;

            try {

                int offset= document.getLineOffset(line);
                int end= document.getLineOffset(line + length);
                length= end - offset;
                Position p= new Position(offset, length);
                document.addPosition(SEGMENTS, p);
                fContent.add(new Segment(MessageFormat.format(JavaEditorM

            } catch (BadPositionCategoryException x) {
            } catch (BadLocationException x) {
```

Compilers

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

Language support

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

Language/library support

- 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?).

Version control

- 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 (gitlab)

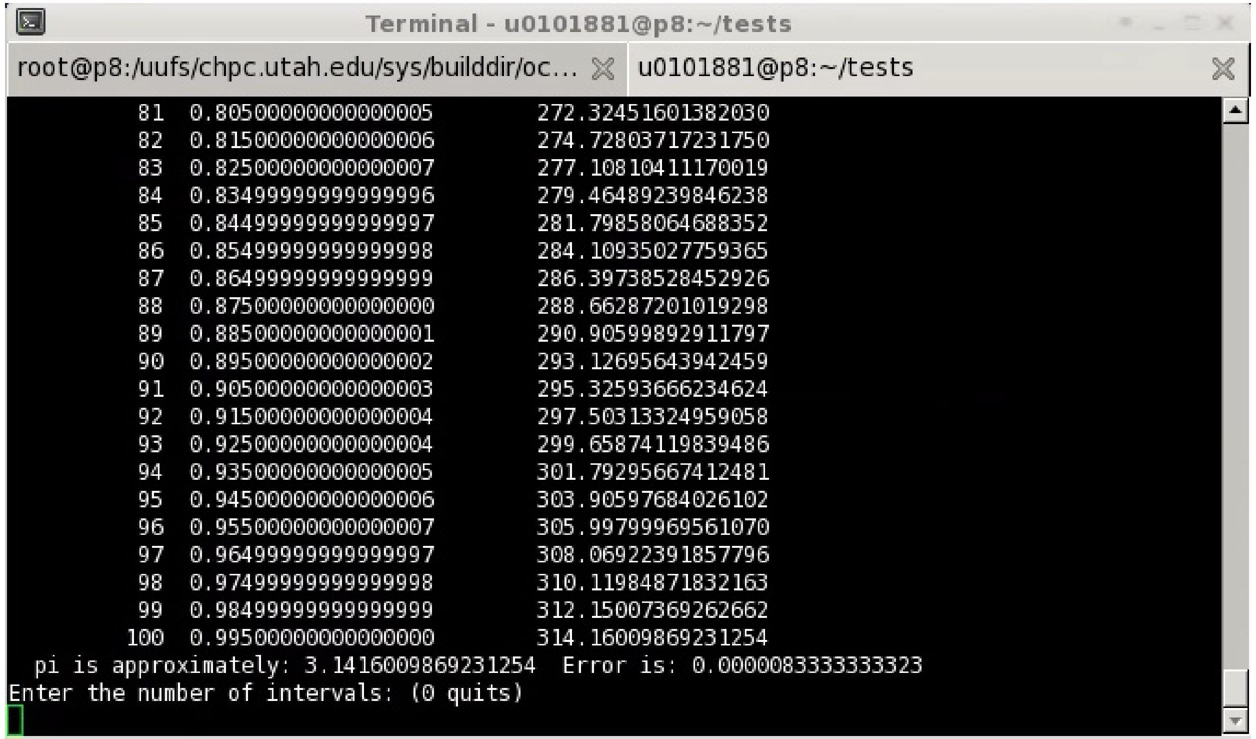
DEBUGGING

Program errors

- 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/printf

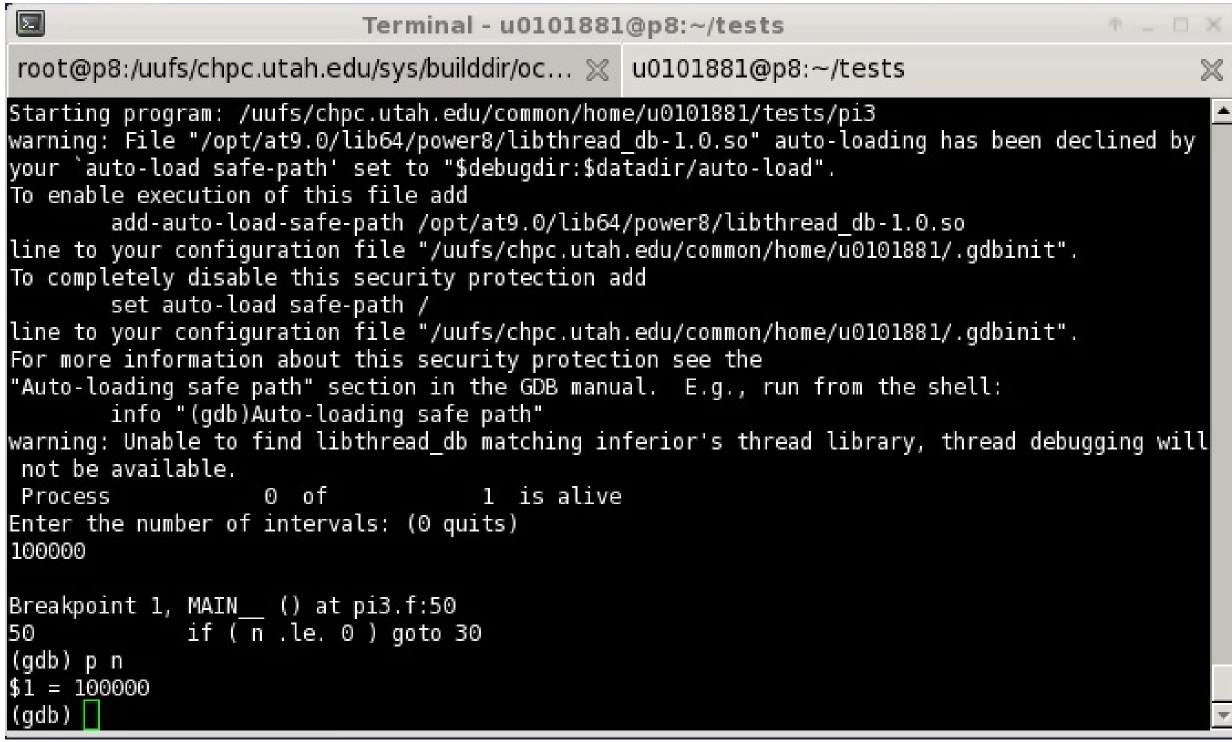
- Write variables of interest into the stdout or file
- Simplest but cumbersome
  - Need to recompile and rerun
  - Need to browse through potentially large output


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

Terminal debuggers

- 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/builddir/oc...
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) 
```

GUI debuggers

- 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

Totalview and DDT

- The only real alternative for parallel or accelerator debugging
- Cost a lot of money (thousands of $), but, worth it
- We had Totalview license (for historical reasons), 32 tokens enough for our needs (renewal ~$1500/yr)
- In 2017 we switched to DDT which gave us competitive upgrade
- XSEDE systems have DDT

How to use Totalview/DDT

1. Compile binary with debugging information

- flag -g

```
gcc –g test.f –o test
```

2. Load module and run Totalview or DDT

```
module load totalview ; module load ddt
```

- TV/DDT + executable

```
totalview executable ; ddt executable
```

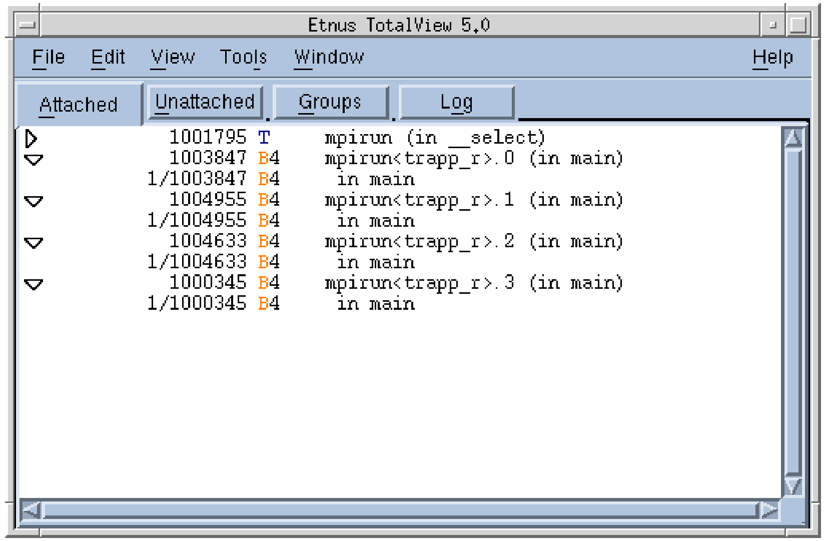
- TV/DDT + core file

```
totalview executable core_file ; ddt executable corefile
```

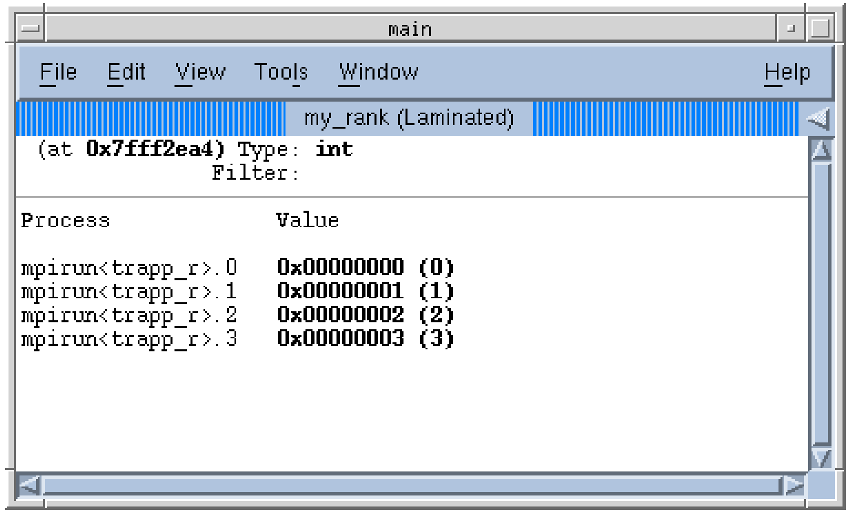
- Run TV/DDT and choose what to debug in a startup dialog

```
totalview ; ddt
```

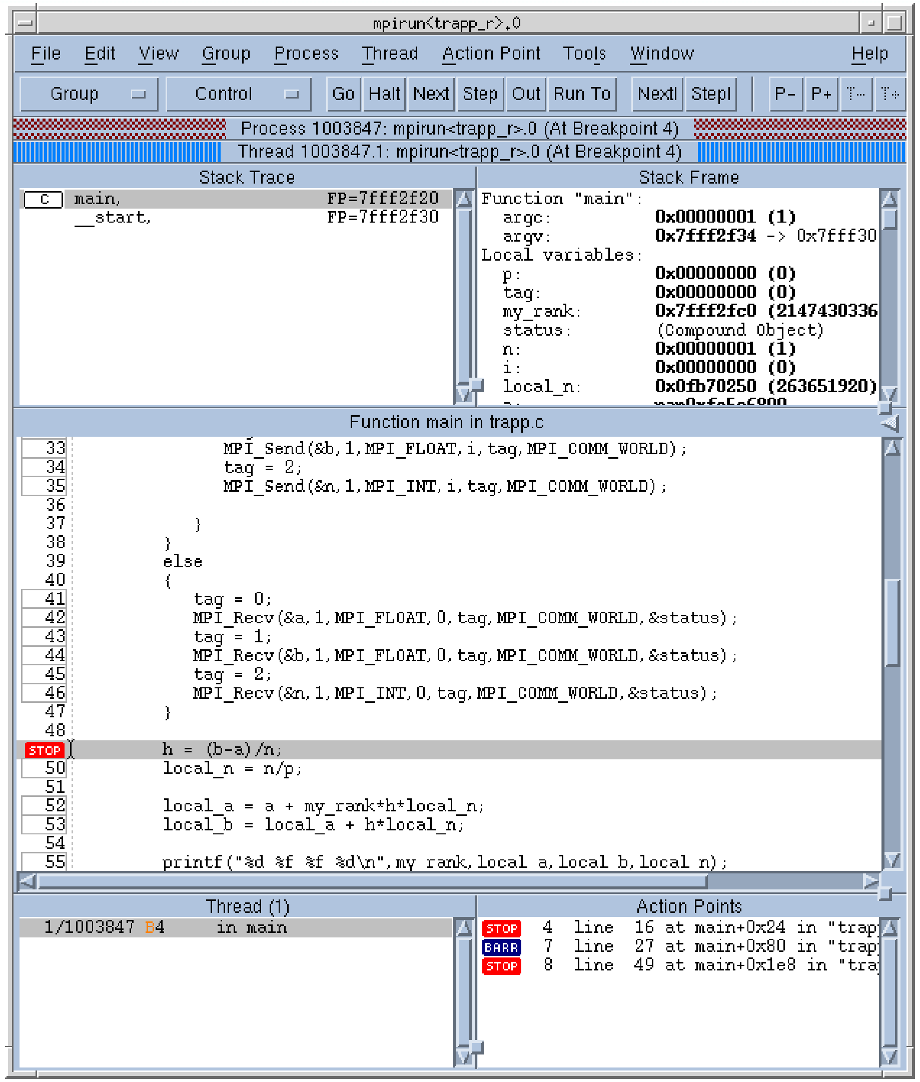
Totalview windows



```
▷       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
```



```
(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)
```



```
Process 1003847: mpirun<trapp_r>.0 (At Breakpoint 4)
Thread 1003847.1: mpirun<trapp_r>.0 (At Breakpoint 4)
```

```
C  main,                  FP=7fff2f20
   __start,               FP=7fff2f30
```

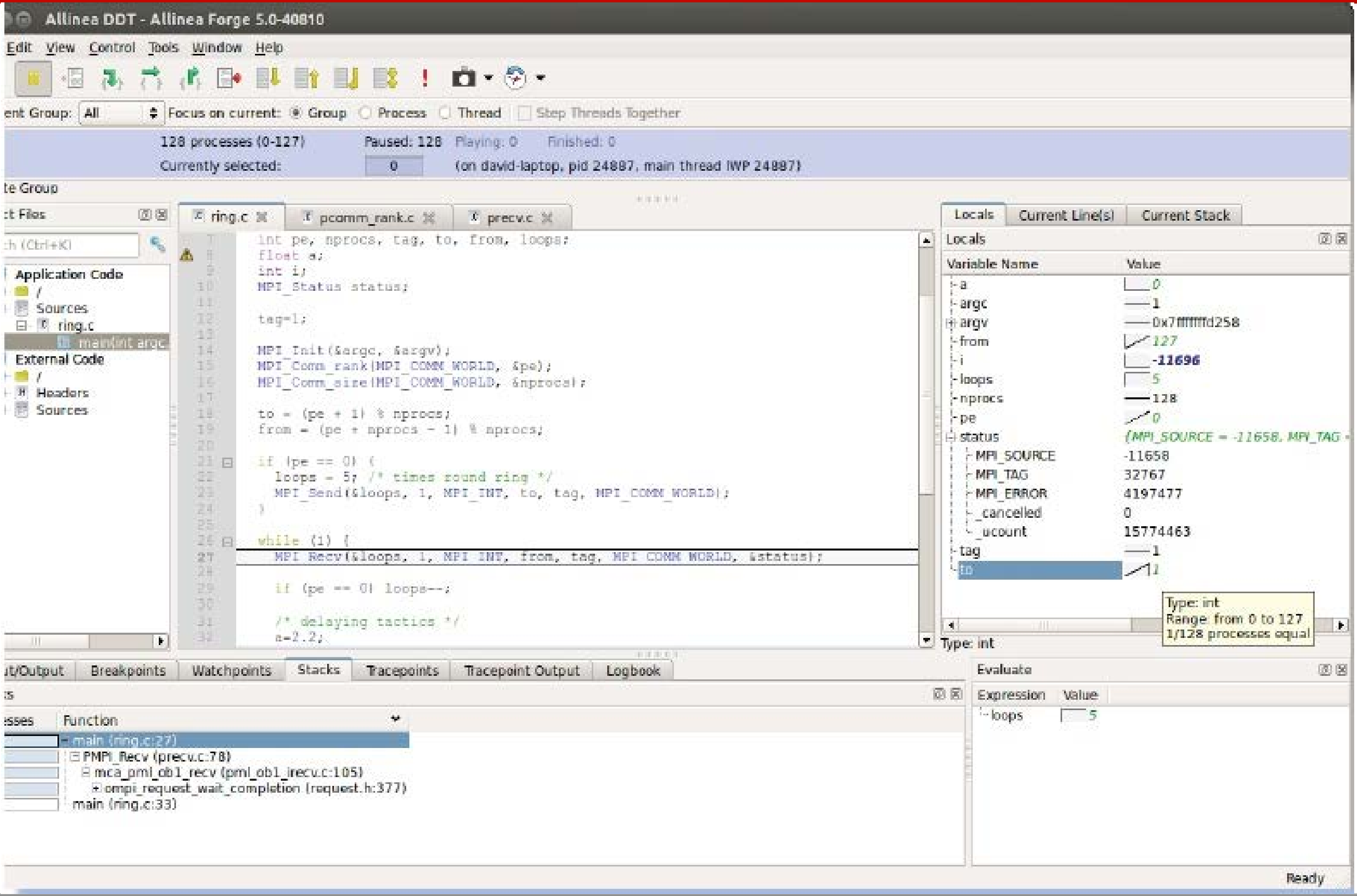
```
Function "main":
  argc:              0x00000001 (1)
  argv:              0x7fff2f34 -> 0x7fff30
Local variables:
  p:                 0x00000000 (0)
  tag:               0x00000000 (0)
  my_rank:           0x7fff2fc0 (2147430336
  status:            (Compound Object)
  n:                 0x00000001 (1)
  i:                 0x00000000 (0)
  local_n:           0x0fb70250 (263651920)
```

```
  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
STOP           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);
```

```
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
```

DDT screenshot



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

```
int pe, nprocs, tag, to, from, loops;
float a;
int i;
MPI_Status status;

tag=1;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &pe);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

to = (pe + 1) % nprocs;
from = (pe + nprocs - 1) % nprocs;

if (pe == 0) {
  loops = 5; /* times round ring */
  MPI_Send(&loops, 1, MPI_INT, to, tag, MPI_COMM_WORLD);
}

while (1) {
  MPI_Recv(&loops, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);

  if (pe == 0) loops--;

  /* delaying tactics */
  a=2.2;
```

Locals

Variable Name	Value
a	0
argc	1
argv	0x7fffffffd258
from	127
i	-11696
loops	5
nprocs	128
pe	0
status	{MPI_SOURCE = -11658, MPI_TAG =
MPI_SOURCE	-11658
MPI_TAG	32767
MPI_ERROR	4197477
_cancelled	0
_ucount	15774463
tag	1
to	1

Type: int
Range: from 0 to 127
1/128 processes equal

Type: int

Expression	Value
loops	5

```
sses  Function
      main (ring.c:27)
        PMPI_Recv (precv.c:78)
          mca_pml_ob1_recv (pml_ob1_irecv.c:105)
            ompi_request_wait_completion (request.h:377)
      main (ring.c:33)
```

Debugger basic operations

- 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

Multiprocess debugging

- 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

Code checkers

- 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

Intel software development products

- We have a 2 concurrent user license + 2 just for compilers
  - One license locks all the tools
  - Cost ~$4000/year + ~$1000 for the compilers
  - Free for students, open source developers, educators
- Tools for all stages of development
  - Compilers and libraries
  - Verification tools
  - Profilers
- More info - https://software.intel.com/en-us/intel-parallel-studio-xe

Intel Inspector

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

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

Intel Inspector

- Source the environment

```
module load inspectorxe
```

- Compile with `–tcheck -g`

```
ifort -openmp -tcheck -g trap.f
```

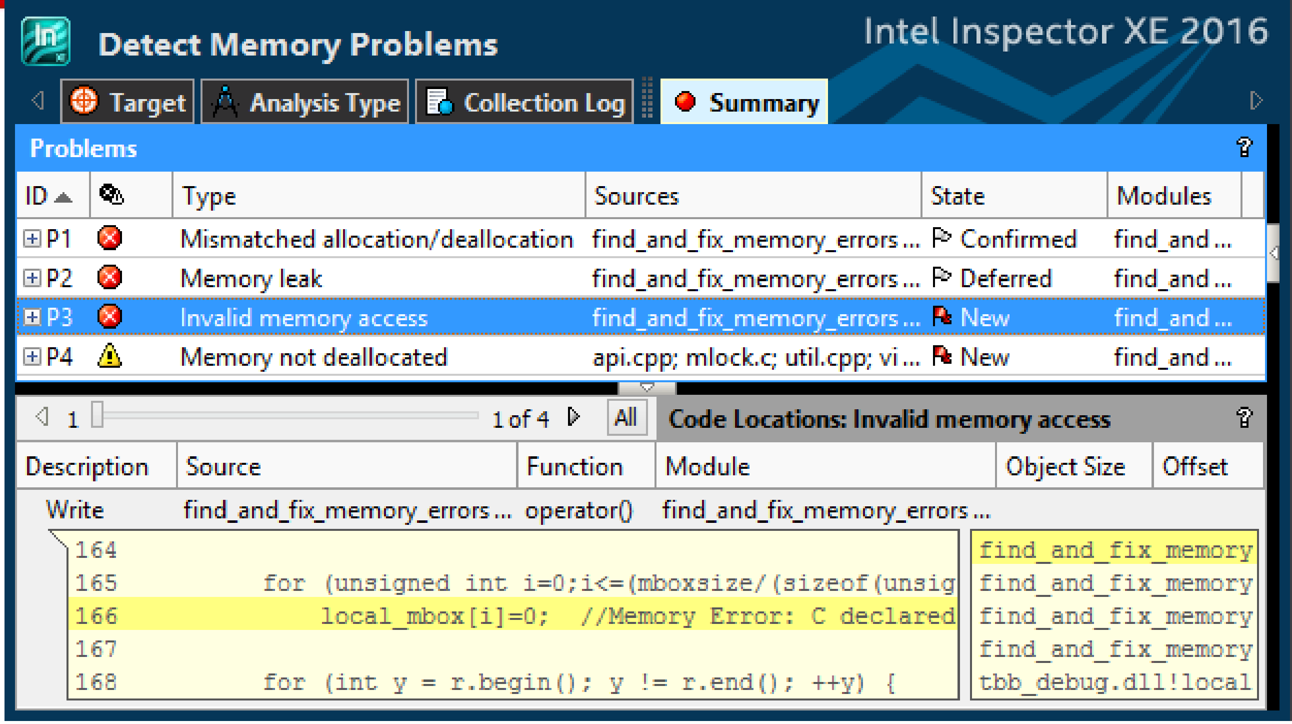
- Run tcheck

`inspxe-gui` – graphical user interface

`inspxe-cl` – command line

- Tutorial

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



Problems

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 ...

1   1 of 4   All

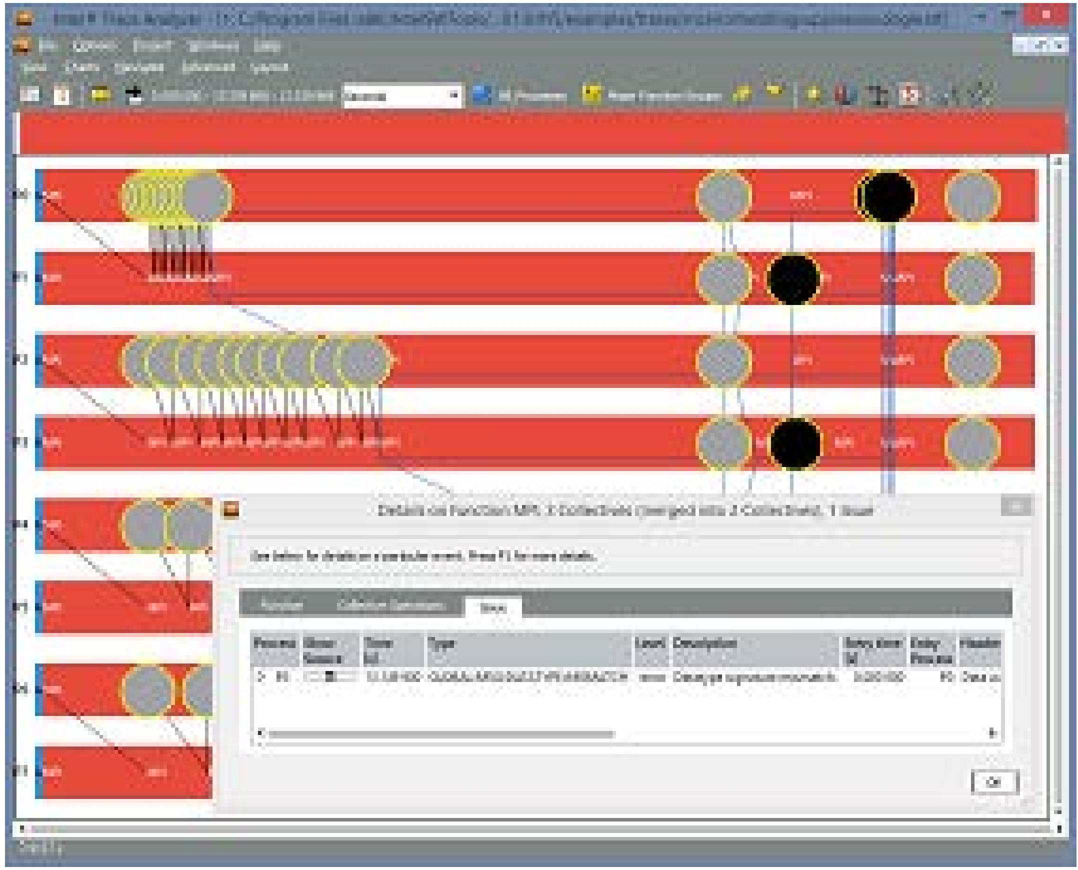
Code Locations: Invalid memory access

Description	Source	Function	Module	Object Size	Offset
Write	find_and_fix_memory_errors ...	operator()	find_and_fix_memory_errors ...		

```

        for (unsigned int i=0;i<=(mboxsize/(sizeof(unsig
            local_mbox[i]=0;   //Memory Error: C declared

        for (int y = r.begin(); y != r.end(); ++y) {
```

```
find_and_fix_memory
find_and_fix_memory
find_and_fix_memory
find_and_fix_memory
tbb_debug.dll!local
```

Intel Trace Analyzer and Collector



- 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

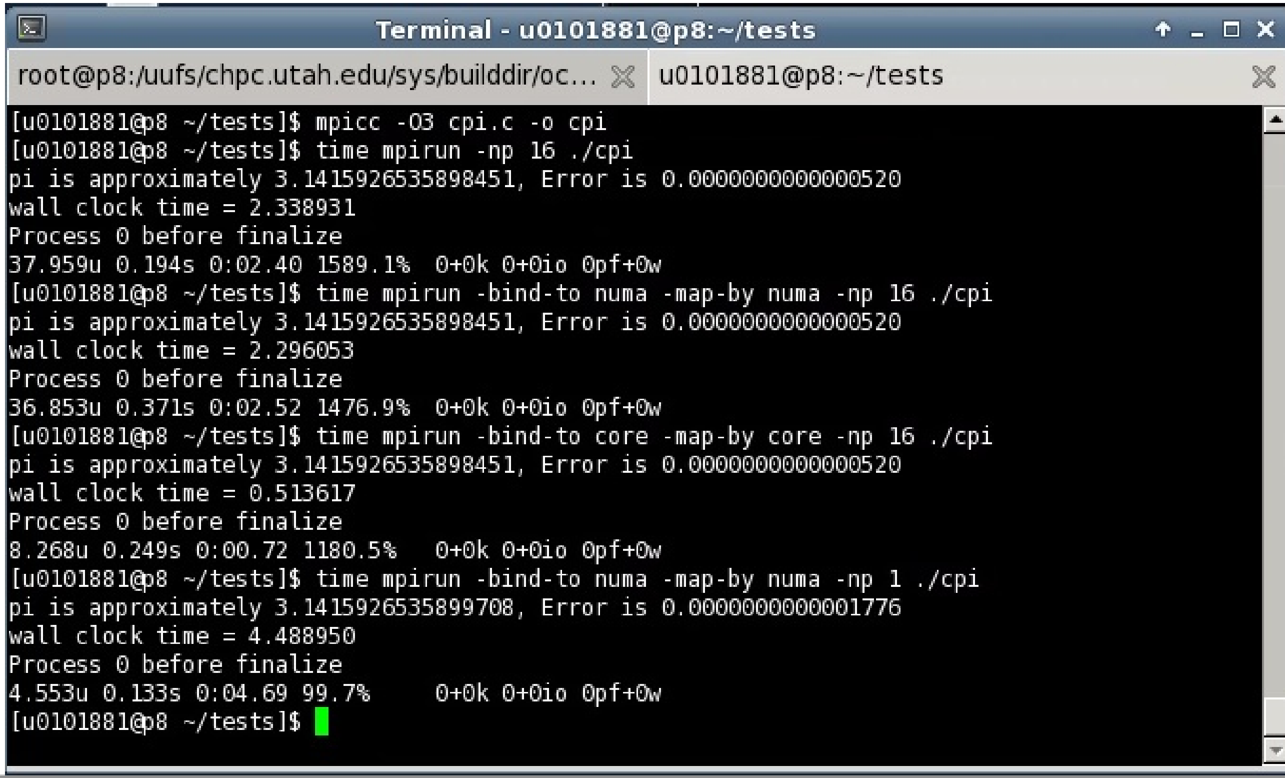
PROFILING

Why to profile

- Evaluate performance
- Find the performance bottlenecks
  - Inefficient programming
    - Array data access, optimized functions, vectorization
  - Memory or I/O bottlenecks
  - Parallel scaling
    - Inefficient parallel decomposition, communication

Program runtime

- 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/builddir/oc…
u0101881@p8:~/tests

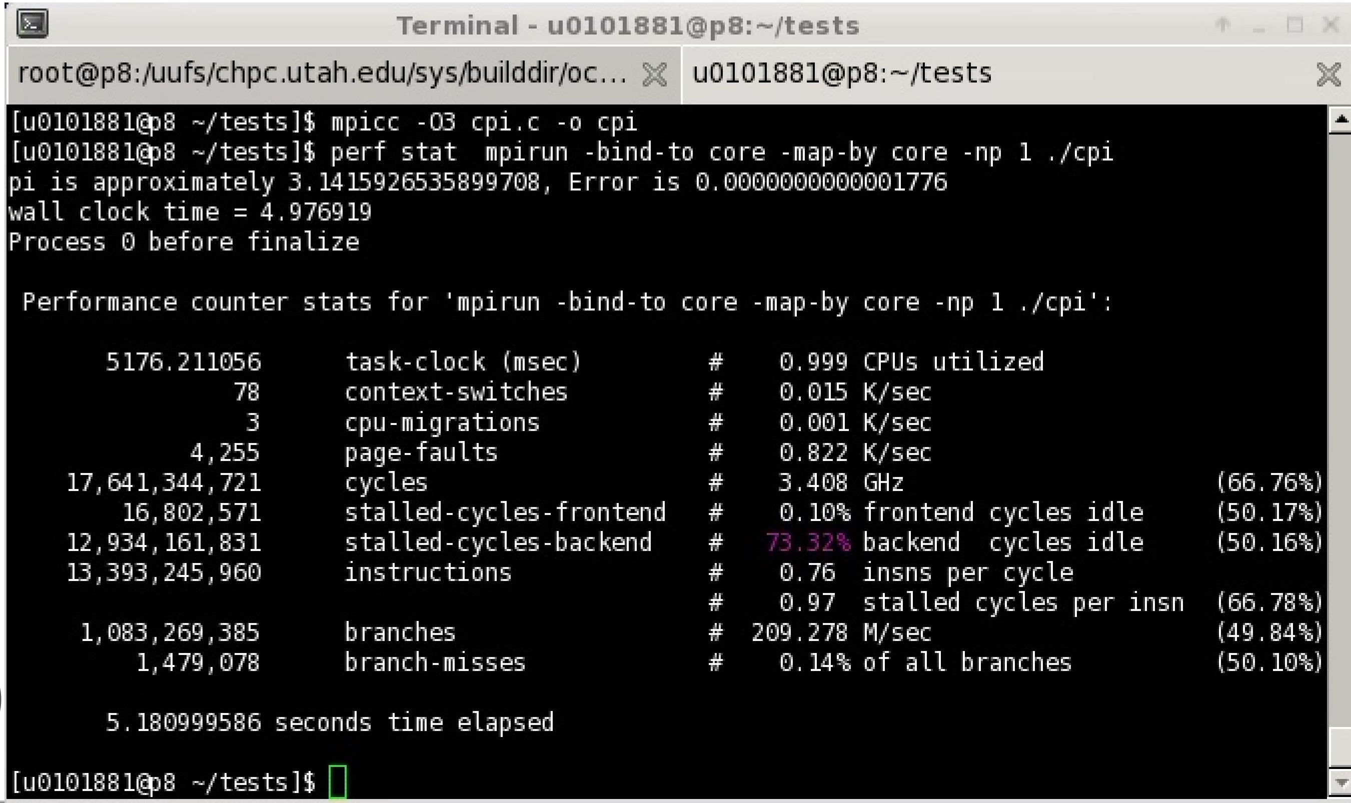
```
[u0101881@p8 ~/tests]$ mpicc -O3 cpi.c -o cpi
[u0101881@p8 ~/tests]$ time mpirun -np 16 ./cpi
pi is approximately 3.1415926535898451, Error is 0.0000000000000520
wall clock time = 2.338931
Process 0 before finalize
37.959u 0.194s 0:02.40 1589.1%  0+0k 0+0io 0pf+0w
[u0101881@p8 ~/tests]$ time mpirun -bind-to numa -map-by numa -np 16 ./cpi
pi is approximately 3.1415926535898451, Error is 0.0000000000000520
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 0.0000000000000520
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]$ 
```

Profiling categories

- 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

Hardware counters

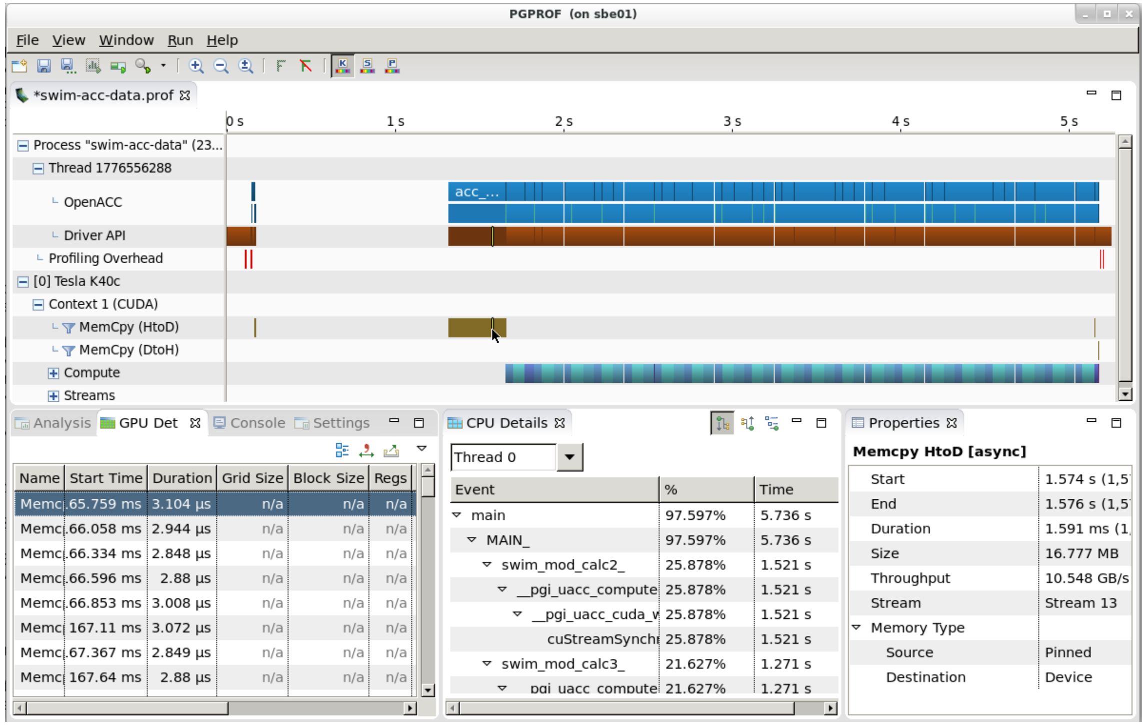
- CPUs include counters to count important events
  - Flops, instructions, cache/memory access
  - Access through kernel or PAPI (Performance Application Programming Interface)
- Tools to analyze the counters
  - perf - hardware counter collection, part of Linux
  - oprofile – profiler + hw counters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


Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/builddir/oc...
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]$ 
```

Serial profiling

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



Name	Start Time	Duration	Grid Size	Block Size	Regs
Memc	.65.759 ms	3.104 µs	n/a	n/a	n/a
Memc	.66.058 ms	2.944 µs	n/a	n/a	n/a
Memc	.66.334 ms	2.848 µs	n/a	n/a	n/a
Memc	.66.596 ms	2.88 µs	n/a	n/a	n/a
Memc	.66.853 ms	3.008 µs	n/a	n/a	n/a
Memc	167.11 ms	3.072 µs	n/a	n/a	n/a
Memc	.67.367 ms	2.849 µs	n/a	n/a	n/a
Memc	167.64 ms	2.88 µs	n/a	n/a	n/a

Thread 0

Event	%	Time
main	97.597%	5.736 s
MAIN_	97.597%	5.736 s
swim_mod_calc2_	25.878%	1.521 s
__pgi_uacc_compute	25.878%	1.521 s
__pgi_uacc_cuda_w	25.878%	1.521 s
cuStreamSynch	25.878%	1.521 s
swim_mod_calc3_	21.627%	1.271 s
pgi_uacc_compute	21.627%	1.271 s

Memcpy HtoD [async]

Start	1.574 s (1,5
End	1.576 s (1,5
Duration	1.591 ms (1,
Size	16.777 MB
Throughput	10.548 GB/s
Stream	Stream 13
Memory Type	
Source	Pinned
Destination	Device

HPC open source tools

- 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
- Score-P/Scalasca
  - Developed by European consortium, did not try yet

Intel tools

- Intel Parallel Studio XE 2017 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)

Intel VTune Amplifier

- 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`

Intel VTune Amplifier

- Source the environment

  `module load vtune`

- Run VTune

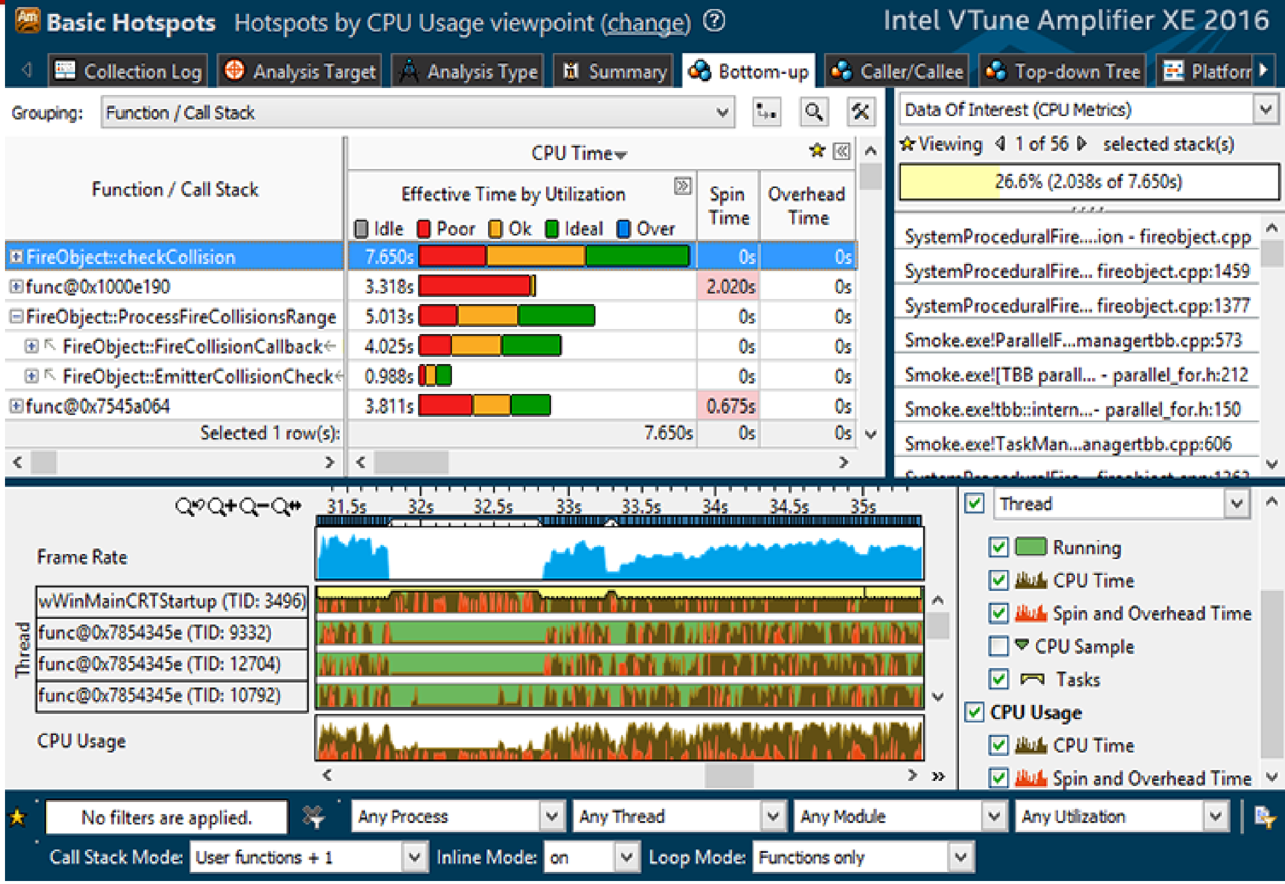
  `amplxe-gui` – GUI

  `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`



Basic Hotspots Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2016

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (🟥 Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	Spin Time	Overhead Time
FireObject::checkCollision	7.650s	0s	0s
func@0x1000e190	3.318s	2.020s	0s
FireObject::ProcessFireCollisionsRange	5.013s	0s	0s
FireObject::FireCollisionCallback	4.025s	0s	0s
FireObject::EmitterCollisionCheck	0.988s	0s	0s
func@0x7545a064	3.811s	0.675s	0s
Selected 1 row(s):	7.650s	0s	0s

Data Of Interest (CPU Metrics)

Viewing 1 of 56 selected stack(s)

26.6% (2.038s of 7.650s)

SystemProceduralFire....ion - fireobject.cpp
SystemProceduralFire... fireobject.cpp:1459
SystemProceduralFire... fireobject.cpp:1377
Smoke.exe!ParallelF...managertbb.cpp:573
Smoke.exe![TBB parall... - parallel_for.h:212
Smoke.exe!tbb::intern...- parallel_for.h:150
Smoke.exe!TaskMan...anagertbb.cpp:606

No filters are applied. | Any Process | Any Thread | Any Module | Any Utilization

Call Stack Mode: User functions + 1 | Inline Mode: on | Loop Mode: Functions only

Intel Advisor

- Vectorization advisor
  - Identify loops that benefit from vectorization, find what is blocking efficient vectorization
  - Useful for speeding up loop performance
- Thread design and prototyping
  - Analyze, design, tune and check threading design
  - Useful for implementing OpenMP in serial code
- More info

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

Intel Advisor

- Source the environment

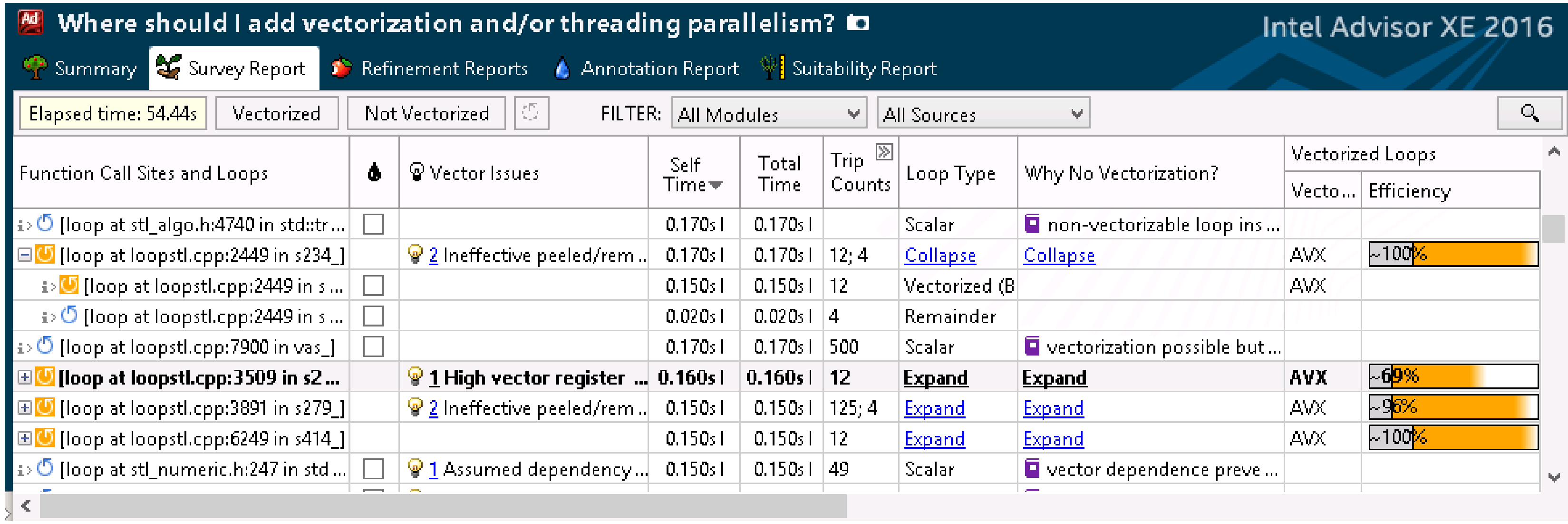
  `module load advisorxe`

- Run Advisor

  `advixe-gui` – GUI

  `advixe-cl` – CLI

- Create project and choose appropriate modeling



Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Elapsed time: 54.44s

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 ...		

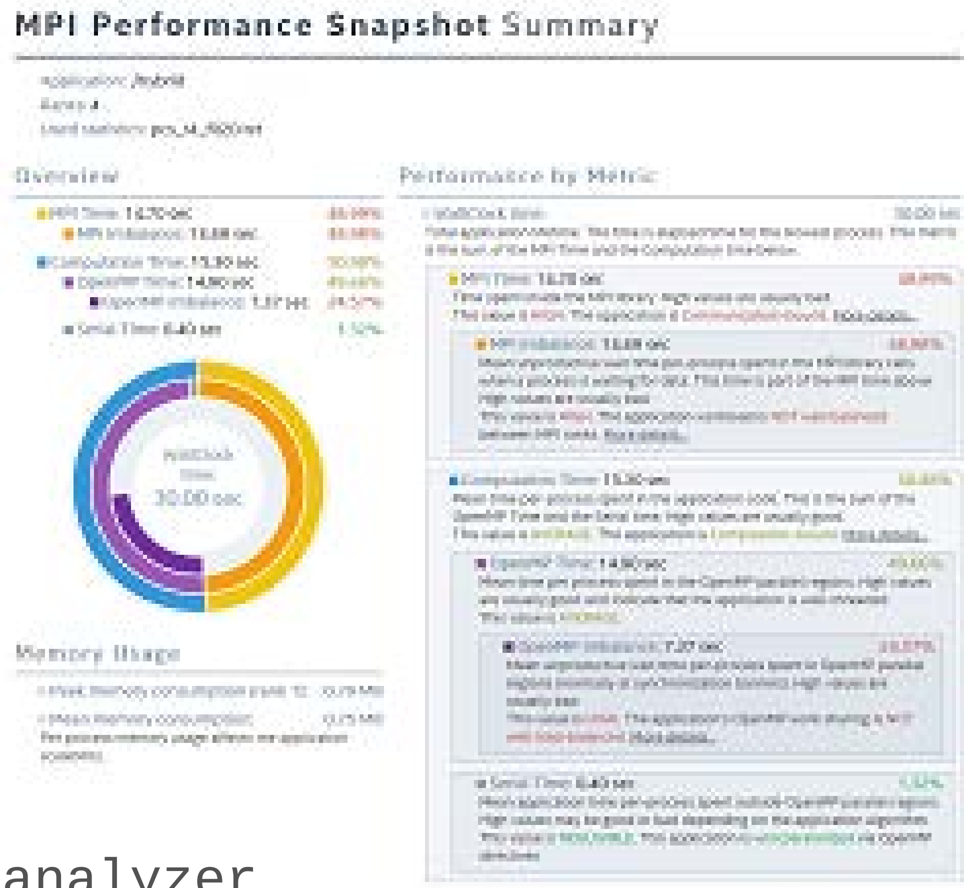
- Getting started guide

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

Intel Trace Analyzer and Collector

- 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

Application: [illegible]

Ranks: 4

[illegible]

Overview

MPI Time: [illegible]

MPI Imbalance: [illegible]

Computation Time: [illegible]

OpenMP Time: [illegible]

OpenMP Imbalance: [illegible]

Serial Time: [illegible]

Memory Usage

[illegible]

Performance by Metric

[illegible]

Intel TAC

- Source the environment

```
module load itac
```

- Using Intel compilers, can compile with –trace

```
mpiifort -openmp –trace trap.f
```

- Run MPI code

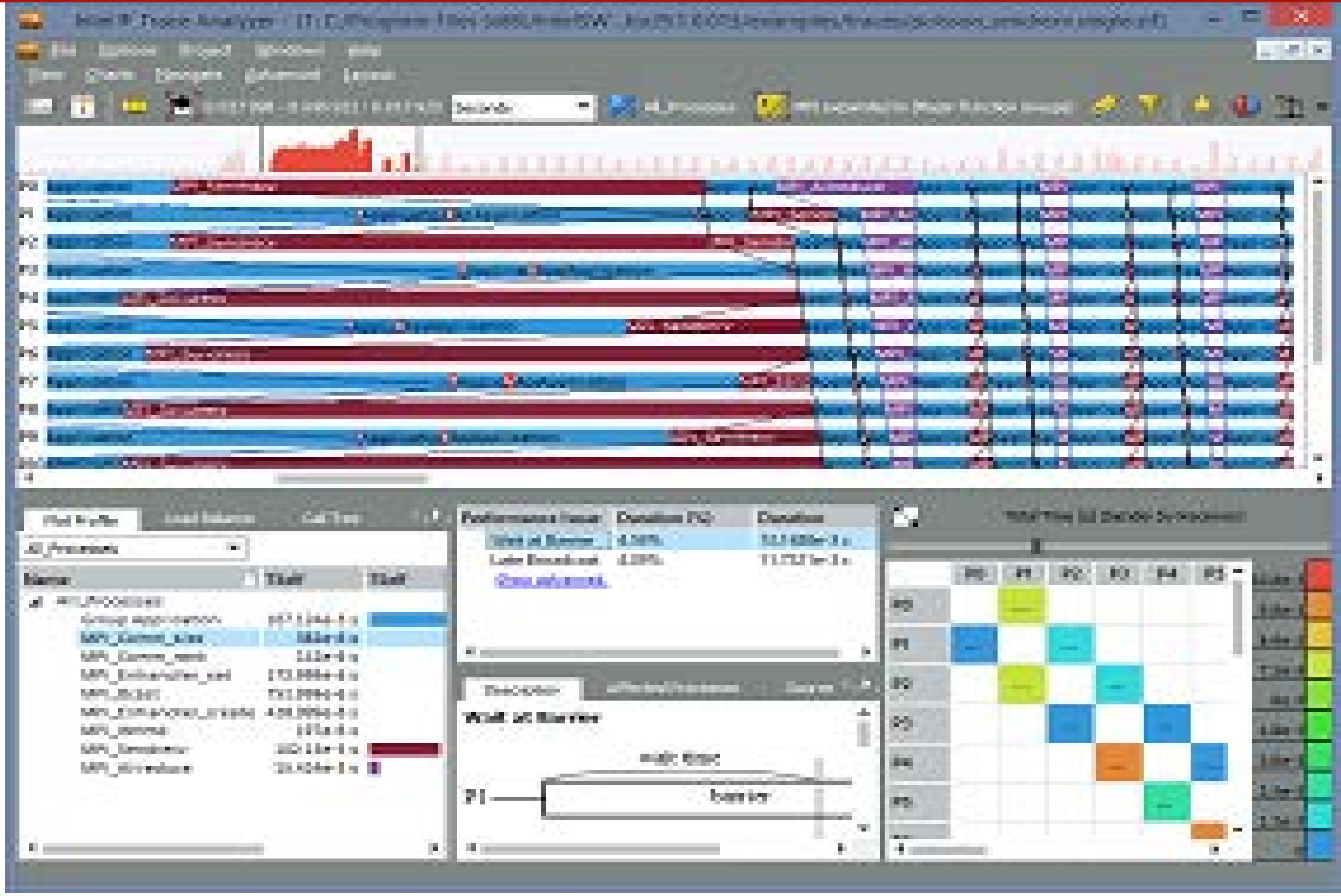
```
mpirun –trace –n 4 ./a.out
```

- Run visualizer

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

- Getting started guide

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



RUNTIME MONITORING

Why runtime monitoring?

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

Runtime monitoring

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

BENCHMARKING

Why to benchmark?

- 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

New system evaluation

- 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

New system evaluation

- 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

Cluster deployment

- 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)

WHAT ELSE DO YOU DO AT YOUR SITE?

BACKUP

Demos

- Totalview
- Advisor
- Inspector
- VTune