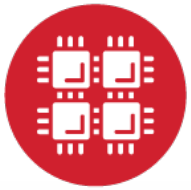


Ohio Supercomputer Center

An OH·TECH Consortium Member

SUG Breakout: Profiling and Debugging Tools

Dr. Judy Gardiner
Dr. Kevin Manalo
April 6, 2017

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

  Resources → Available Software
- Contact the help desk (OSC Help)

  oschelp@osc.edu

  614-292-1800

  1-800-686-6472
- Watch for training courses to be offered this summer

Profiling / Debugging Tools Available at OSC

- Debugging tools
  - TotalView
  - Allinea DDT (trial license only)
- Profiling tools
  - Allinea Performance Reports
  - Allinea MAP
  - Intel VTune
  - Intel Trace Analyzer and Collector (ITAC)
  - Intel Advisor

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
  - Example: **`icc -g -o mycode mycode.c`**
  - Example: **`icc -g -O0 -o mycode mycode.c`**

TotalView debugger

- Available on all OSC clusters
  - `module load totalview`
- For a non-MPI program:
  - `totalview ./mycode [-a prog_args]`
  - `totalview ./setsm -a file1 file1 -outres 8`
- For an MPI program:
  - `totalview -args mpiexec [mpiexec arguments] ./mycode [program arguments]`
  - `totalview -args mpiexec -ppn 1 ./mycode`

Allinea DDT

- Trial license - all OSC clusters (hidden module)
  - `module load allinea/.testDDT-7.0`
  - `ddt ./mycode`

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 -O*n*
  - Example: **`icc -g -O3 -o mycode mycode.c`**

- Use the same level of optimization you normally do
  - Bad example: **`icc -g -o mycode mycode.c`**
    - Equivalent to -O0

Allinea Performance Reports

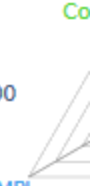
- Easy to use
  - "-g" flag not needed - works on precompiled binaries
- Gives a summary of your code's performance
  - view report with browser
- For a non-MPI program:
  - `module load allinea`
  - `perf-report --no-mpi ./mycode [`*`args`*`]`
- For an MPI program:
  - `perf-report -np` *`num_procs`*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

setsm - Performance Report

file:///C:/Users/judithg/Documents/Training/SUG breakout/setsm_1p_2015-08-28_11-03.html 67%

allinea PERFORMANCE REPORTS

Command:	/nfs/14/judithg/howat/SETSM_c_0224/setsm /fs/lustre/osu7360 /data/WV01_20150508_102001003EA5DA00_102001003C02A600 /WV01_15MAY080613301-P1BS-102001003C02A600.tif /fs/lustre/osu7360 /data/WV01_20150508_102001003EA5DA00_102001003C02A600 /WV01_15MAY080614188-P1BS-102001003EA5DA00.tif ../Results/pbs_194942
Resources:	1 node (20 physical, 20 logical cores per node)
Memory:	63 GB per node
Tasks:	1 process, OMP_NUM_THREADS was 0
Machine:	r0019.ten.osc.edu
Start time:	Fri Aug 28 11:03:22 2015
Total time:	667 seconds (11 minutes)
Full path:	/nfs/14/judithg/howat/SETSM_c_0224
Input file:	
Notes:	



Summary: setsm is Compute-bound in this configuration

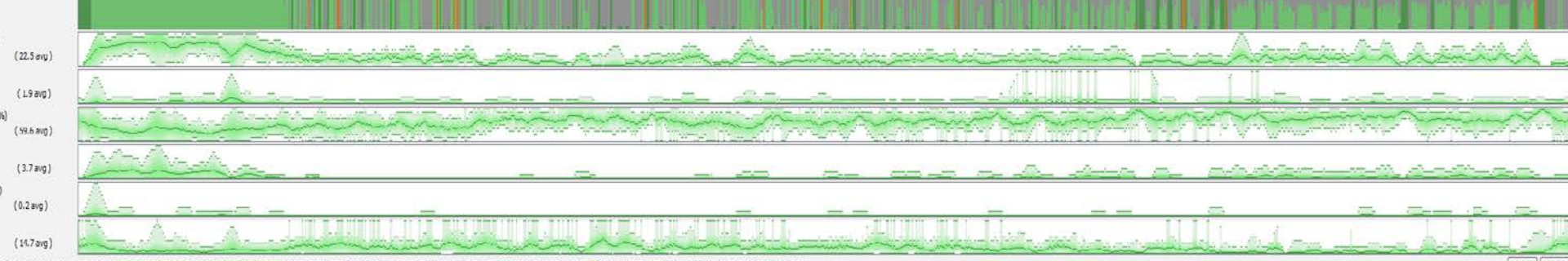
Compute	96.0%	Time spent running application code. High values are usually go This is **very high**; check the CPU performance section for advice
MPI	0.0%	Time spent in MPI calls. High values are usually bad. This is **very low**; this code may benefit from a higher process co
I/O	4.0%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wa

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

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

Allinea MAP

- Interpretation of profile requires some expertise
- Gives details about your code's performance
- For a non-MPI program:
  - `module load allinea`
  - `map --profile --no-mpi ./mycode [`*`args`*`]`
- For an MPI program:
  - `map --profile -np` *`num_procs`* `./mycode [`*`args`*`]`
- View and explore resulting profile using Allinea client



ss): Main thread compute 9 %, OpenMP 49 %, File I/O 2 %, OpenMP overhead 40 %, Sleeping 0 % | CPU floating-point **22.5** %; CPU integer **1.9** %; CPU memory access **59.6** %; CPU fp vector **3.7** %; CPU integer vector **0.2** %; CPU branch **14.7** %;

Metrics... Select A...

```
1100
1101
1102                                        //printf("play %d\n",lower_level_match);
1103                                    }
1104                                    else
1105                                    {
1106                                        count blunder = DecisionMPs(true,count MPs,subBoundary,GridPT3,level,grid resolution,iteration,Size Grid2D,filename mps_pre,filename_mps,proinfo.save_filepath,
1107                                                                    Hinterval,&lower level match,&prc 3sigma,&prc mean,count results,&minH mps,&maxH mps,minmaxHeight,
1108                                                                    SubMagImages_L,SubMagImages_R,grid_resolution, Image_res[0],LRPCs,RRPCs,
1109                                                                    Limagesize,data_size_l[level],SubImages_L,Rimagesize,data_size_r[level],SubImages_R,Template_size,
1110                                                                    param, Grid wgs,GridPT,
1111                                                                    NumOfIAparam, t_Rimageparam, Lstartpos, Rstartpos,
1112                                                                    proinfo.save_filepath,row,col,proinfo.pre_DEMtif);
1113                                        count MPs              = count results[0];
1114                                        //printf("play %d\n",lower_level_match);
1115                                        printf("row = %d\tcol = %d\tlevel = %d\titeration = %d\tEnd blunder points\n" row col level iteration);
```

OpenMP Stacks | OpenMP Regions | Functions

Total	Child	Overhead	Function
16.3%		16.3%	__kmp_launch_worker(void*)
16.3%		16.3%	__kmp_launch_thread
16.3%		16.3%	wait (OpenMP Overhead)
16.3%		16.3%	__kmp_fork_barrier(int, int)
16.3%		16.3%	__kmp_hyper_barrier_release(barrier_type, kmp_info*, int, int, int, void*) (OpenMP Overhead)
15.0%	0.1%	0.1%	VerticalLineLocus_blunder [OpenMP region 1]
14.3%		14.3%	__kmp_wait_yield_4
9.2%		9.2%	__kmp_wait_template (OpenMP Overhead)
7.7%		7.7%	suspend [inlined] (OpenMP Overhead)
7.7%		7.7%	__kmp_suspend_64 (OpenMP Overhead)
7.7%		7.7%	__kmp_suspend_template [inlined] (OpenMP Overhead)
7.7%		7.7%	pthread_cond_wait (OpenMP Overhead)
7.1%		7.1%	__kmp_wait_template [inlined] (OpenMP Overhead)
6.2%		6.2%	__kmp_dispatch_next
6.7%		6.7%	test_then_inc_acq [inlined]
5.4%			exp.L
10.0%	5.8%		Orientation [inlined]
7.9%	4.4%		GetObjectToImageRPC_single
3.6%	0.7%		VerticalLineLocus [OpenMP region 1]
2.9%		2.9%	__kmp_x86_pause [inlined]
3.3%	0.6%		malloc
4.5%	2.3%		_IO_vfscanf_internal
2.0%			__ll_lock_wait_private
1.7%		1.7%	__kmp_wait_yield_4
1.7%	<0.1%	<0.1%	SetHeightRange_blunder [OpenMP region 0]
1.6%			__rinse_norannel
1.5%		1.5%	sched_yield
1.5%			sin.N
5.0%	3.6%		ps2xys_single [inlined]
1.4%		1.4%	__kmp_x86_pause [inlined] (OpenMP Overhead)
1.7%		1.7%	__kmp_xq_4
1.0%			_int_free
1.0%			::CreateImagePyramid(void) [OpenMP region 1]
1.5%	0.6%		___strtof_l_internal
0.8%			pow.L
0.8%		0.8%	__kmp_launch_monitor(void*)
17.4%	16.7%	5.0%	VerticalLineLocus_blunder (OpenMP)
0.7%			LZWDecode
0.7%			atan2
0.6%			__mpn_divrem
0.6%			_int_malloc
0.6%		0.6%	sched_yield (OpenMP Overhead)

Intel VTune

- A profiler that can work on 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`

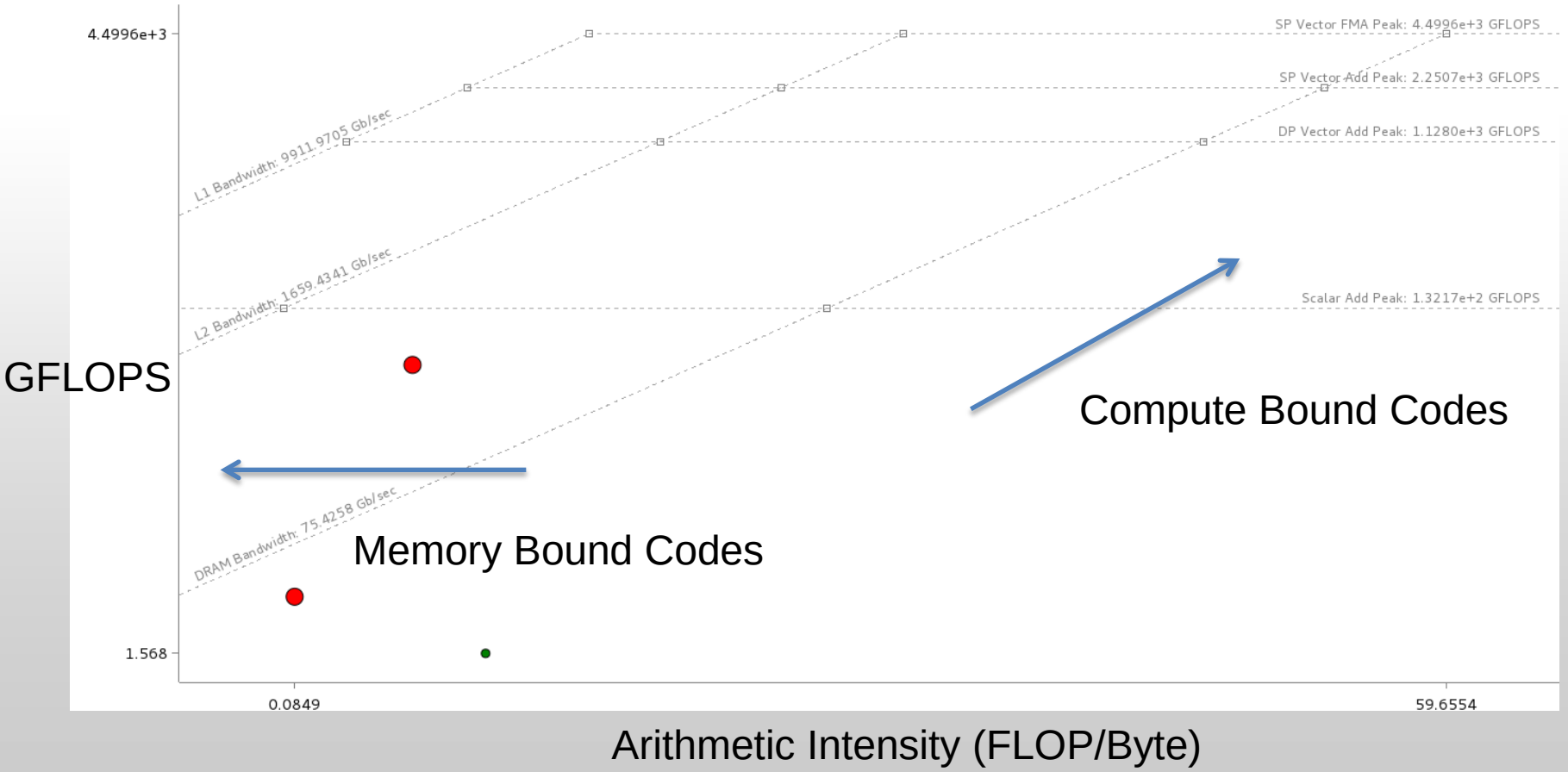
Intel Trace Analyzer and Collector (ITAC)

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

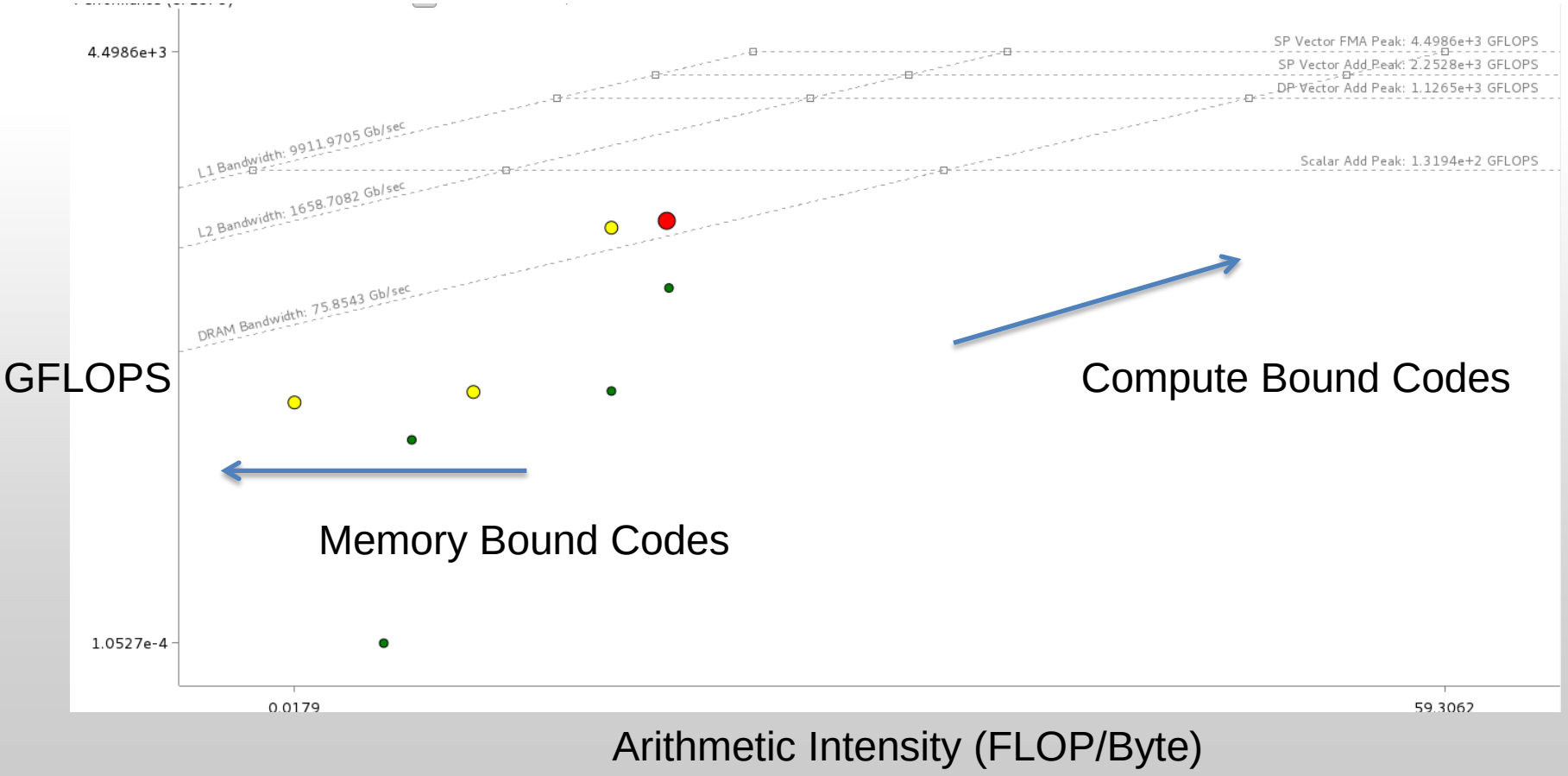
Intel Advisor

- 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`

Intel Advisor (Roofline Chart Before Optimization)



Intel Advisor (Roofline Chart After Optimization)



Questions

Judy Gardiner
Scientific Applications Engineer
Ohio Supercomputer Center
judithg@osc.edu

1224 Kinnear Road
Columbus, OH 43212
Phone: (614) 292-9623

ohiosupercomputercenter

ohiosupercomputerctr