Performance Bottleneck(s)

Jerome Vienne
viennej@tacc.utexas.edu

Texas Advanced Computing Center

Application Performance Snapshot

Take a quick look at your application's performance to see if it is well optimized for modern hardware.

- MPI parallelism (Linux* only)
- OpenMP* parallelism
- Memory access
- FPU Utilization
- I/O efficiency

Free !!!

It is easy to install, easy to run and provides results in a text or HTML report.

Website: http://tinyurl.com/aps-knl

Simple to use

- Part of Vtune or can be installed from the web
- You just have to call aps or aps.sh.
  aps.sh ./a.out
  mpirun –np 6 aps ./a.out
- It will generate a text file
- To generate the html you will have to recall it with specific flags (provided at the end of the initial run)

- There will be one exercise that will be provided during the lab

An Overview of Profiling

Outline

Introduction

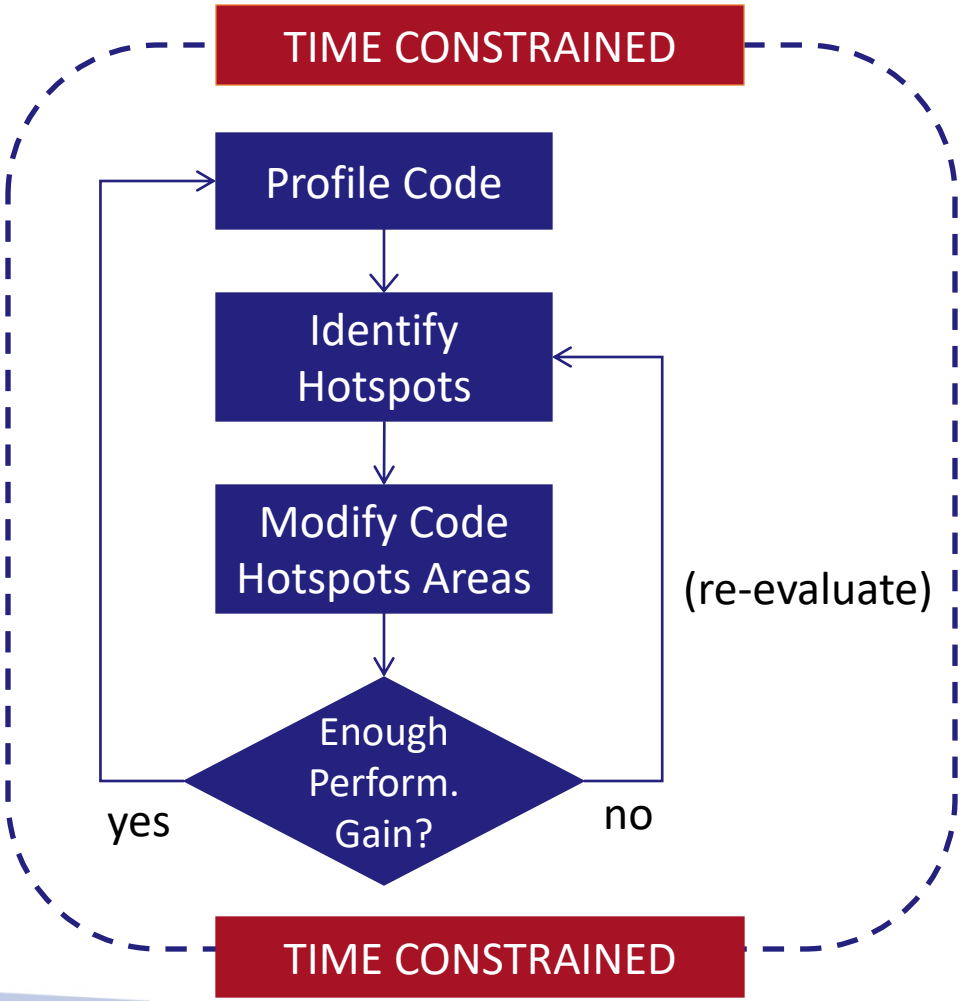
Basic Tools
- Command line timers
- Code timers
- gprof

Other Tools ..

Intermediate tools
- Perfexpert
- IPM

Advanced Tools
- Vtune
- Intel Advisor
- HPCToolkit
- Tau

Optimization Process



- Iterative process
- Application dependent
- Different levels
  - Compiler Options
  - Performance Libraries
  - Code Optimizations

Timers: Command Line

The command **time** is available in most Unix systems.

It is simple to use (no code instrumentation required).

Gives total execution time of a process and all its children in seconds.

```
% /usr/bin/time -p ./exeFile
real 9.95
user 9.86
sys 0.06
```

Leave out the **-p** option to get additional information:

```
% time ./exeFile
% 9.860u 0.060s 0:09.95 99.9%
```

Timers: Code Section

```
  INTEGER :: rate, start, stop
  REAL    :: time

  CALL SYSTEM_CLOCK(COUNT_RATE = rate)
  CALL SYSTEM_CLOCK(COUNT = start)

  ! Code to time here

  CALL SYSTEM_CLOCK(COUNT = stop)
  time = REAL( ( stop - start )/ rate )
```

```
#include <time.h>

double start, stop, time;
start = (double)clock()/CLOCKS_PER_SEC;

 /* Code to time here */

stop = (double)clock()/CLOCKS_PER_SEC;
time = stop - start;
```

About GPROF

GPROF is the **G**NU Project **PROF**iler.
gnu.org/software/binutils/

Requires recompilation of the code.

Compiler options and libraries provide wrappers for each routine call and periodic sampling of the program.

Provides three types of profiles

- Flat profile
- Call graph
- Annotated source

Types of Profiles

Flat Profile

- CPU time spend in each function (self and cumulative)
- Number of times a function is called
- **Useful to identify most expensive routines**

Call Graph

- Number of times a function was called by other functions
- Number of times a function called other functions
- Useful to identify function relations
- Suggests places where function calls could be eliminated

Annotated Source

- Indicates number of times a line was executed

Profiling with gprof

Use the **-pg** flag during compilation:

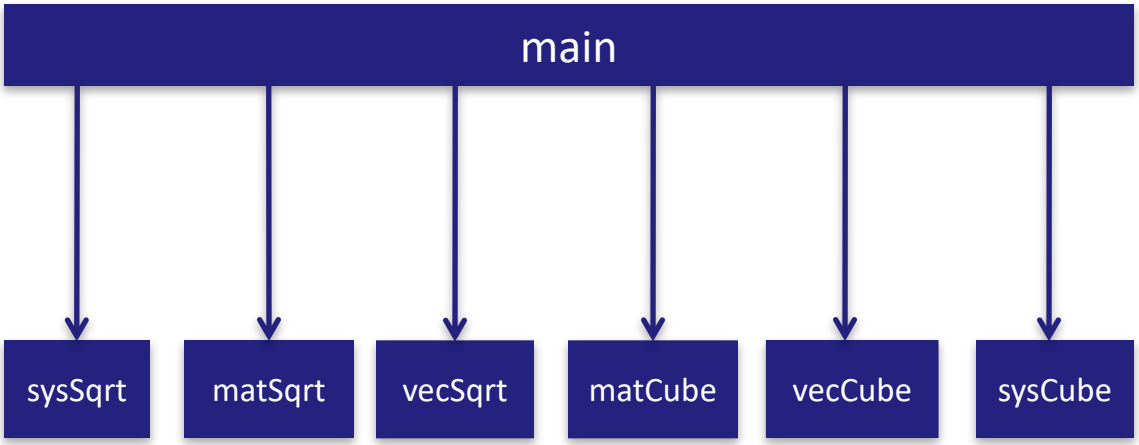
```
% gcc  -g -pg ./srcFile.c

% icc  -g -pg  ./srcFile.c
```

Run the executable. An output file **gmon.out** will be generated with the profiling information.

Execute **gprof** and redirect the output to a file:

```
% gprof    ./exeFile gmon.out > profile.txt

% gprof -l ./exeFile gmon.out > profile_line.txt

% gprof -A ./exeFile gmon.out > profile_anotated.txt
```

Visual Call Graph



Flat profile

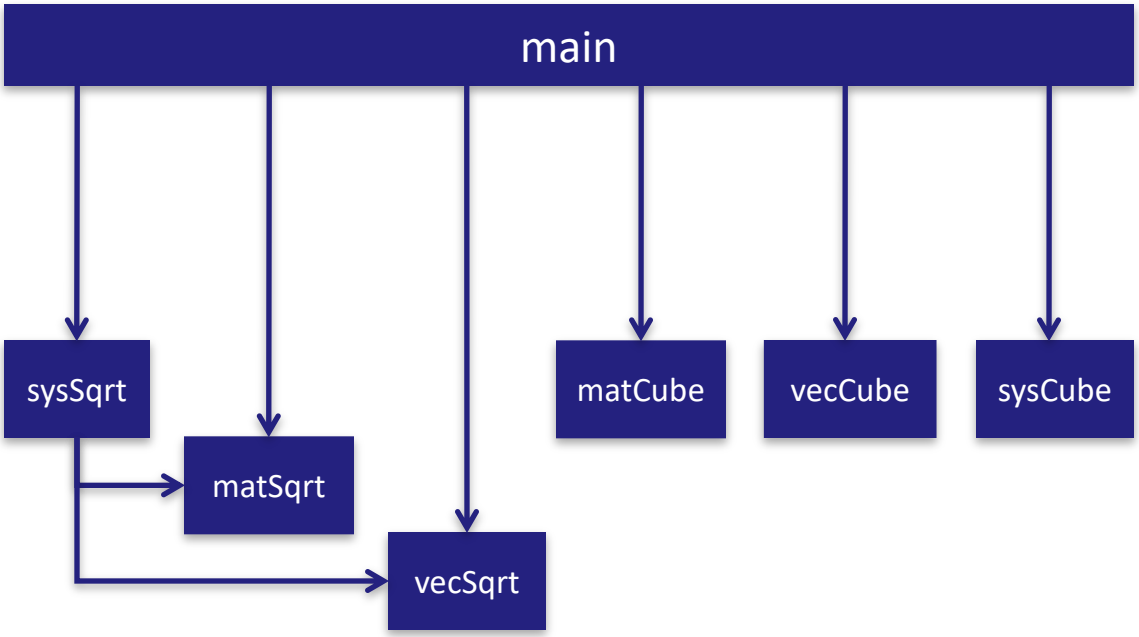
In the flat profile we can identify the most expensive parts of the code (in this case, the calls to matSqrt, matCube, and sysCube).

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
50.00	2.47	2.47	2	1.24	1.24	matSqrt
24.70	3.69	1.22	1	1.22	1.22	matCube
24.70	4.91	1.22	1	1.22	1.22	sysCube
0.61	4.94	0.03	1	0.03	4.94	main
0.00	4.94	0.00	2	0.00	0.00	vecSqrt
0.00	4.94	0.00	1	0.00	1.24	sysSqrt
0.00	4.94	0.00	1	0.00	0.00	vecCube

Call Graph Profile

```
index % time    self  children    called     name
                0.00    0.00       1/1           <hicore> (8)
[1]     100.0   0.03    4.91       1         main [1]
                0.00    1.24       1/1           sysSqrt [3]
                1.24    0.00       1/2           matSqrt [2]
                1.22    0.00       1/1           sysCube [5]
                1.22    0.00       1/1           matCube [4]
                0.00    0.00       1/2           vecSqrt [6]
                0.00    0.00       1/1           vecCube [7]
-----------------------------------------------
                1.24    0.00       1/2           main [1]
                1.24    0.00       1/2           sysSqrt [3]
[2]      50.0   2.47    0.00       2         matSqrt [2]
-----------------------------------------------
                0.00    1.24       1/1           main [1]
[3]      25.0   0.00    1.24       1         sysSqrt [3]
                1.24    0.00       1/2           matSqrt [2]
                0.00    0.00       1/2       2   vecSqrt [6]
-----------------------------------------------
```

Visual Call Graph



PERFEXPERT

Perfexpert

A new tool, locally developed at UT

Easy to use and understand

Great for quick profiling and for beginners

Provides recommendation on “what to fix” in a subroutine

Collects information from PAPI using HPCToolkit

Combines ease of use with useful interpretation of gathered performance data

Optimization suggestions!!!

Profiling with Perfexpert: Compilation

Compile the code with full optimization and with the -g flag:

```
mpicc -g -O3 source.c

mpif90 -g -O3 source.f90
```

In your job submission script:

```
perfexpert 0.1 ./<executable> <executable args>
```

Perfexpert Output

```
Loop in function collision in collision.F90:68 (37.66% of the total runtime)
================================================================================
ratio to total instrns    %  0.........25.........50.........75.........100
 - floating point      100.0 ************************************************
 - data accesses        31.7 ****************
* GFLOPS (% max)        19.8 **********
 - packed                7.3 ****
 - scalar               12.5 ******
--------------------------------------------------------------------------------
performance assessment  LCPI good.......okay........fair........poor........bad
* overall               4.51 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
* data accesses        12.28 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
 - L1d hits             1.11 >>>>>>>>>>>>>>>>>>>>>>>
 - L2d hits             1.05 >>>>>>>>>>>>>>>>>>>>>>
 - L3d hits             0.01
 - LLC misses          10.11 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
* instruction accesses  0.13 >>>
 - L1i hits             0.00
 - L2i hits             0.01
 - L2i misses           0.12 >>
* data TLB              0.02
* instruction TLB       0.00
* branch instructions   0.00
 - correctly predicted  0.00
 - mispredicted         0.00
* floating-point instr  8.01 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
 - slow FP instr        0.16 >>>
 - fast FP instr        7.85 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
================================================================================
```

Perfexpert Suggestions

```
#-------------------------------------------------
# Recommendations for collision.F90:68
#-------------------------------------------------
#
# Here is a possible recommendation for this code segment
#
Description: compute values rather than loading them if doable with few operations
Reason: this optimization replaces (slow) memory accesses with equivalent but faste
computations
Code example:


loop i {
  t[i] = a[i] * 0.5;
}
loop j {
  a[j] = c[j] - t[j];
}
 =====>
loop i {
  a[i] = c[i] - (a[i] * 0.5);
}
```

TACC

Advanced Perfexpert Use

If you select the -m or -s options, PerfExpert will try to

- Automatically optimize your code
- Show the performance analysis report
- Show the list of suggestion for bottleneck remediation when no automatic optimization is possible.

Get the code and documentation from:

www.tacc.utexas.edu/research-development/tacc-projects/perfexpert

github.com/TACC/perfexpert

IPM: INTEGRATED PERFORMANCE MONITORING

IPM: Integrated Performance Monitoring

"IPM is a portable profiling infrastructure for parallel codes. It provides a low-overhead performance summary of the computation and communication in a parallel program"

IPM is a quick, easy and concise profiling tool

The level of detail it reports is smaller than TAU, PAPI or HPCToolkit

IPM: Integrated Performance Monitoring

IPM features:

- easy to use
- has low overhead
- is scalable

Requires no source code modification, just adding the “-g” option to the compilation

Produces XML output that is parsed by scripts to generate browser-readable html pages

IPM: Integrated Performance Monitoring

Define the type of collection:

export IPM_REPORT=full (full, terse, none)

Define collection threshold (optional)

export IPM_MPI_THRESHOLD=0.3

(Reports only routines using this percentage or more of MPI time)

Generating HTML output:

ipm_parse -html <collected data dir>

IPM: Text Output

```
##IPMv0.983############################################################################
#
# command : /home1/01157/carlos/TEST/profiling/mm  (completed)
# host    : c404-504/x86_64_Linux          mpi_tasks : 16 on 1 nodes
# start   : 11/11/14/16:25:04              wallclock : 0.836220 sec
# stop    : 11/11/14/16:25:05              %comm     : 20.49
# gbytes  : 5.25372e+00 total              gflop/sec : NA
#
############################################################################
# region  : *       [ntasks] =     16
#
#                          [total]         <avg>           min           max
# entries                       16             1             1             1
# wallclock                 13.3793      0.836207      0.836197       0.83622
# user                      18.5732       1.16082       1.05884       1.18082
# system                    1.48777     0.0929854      0.074988      0.197969
# mpi                       2.74098      0.171311      0.133593      0.650309
# %comm                                   20.4864       15.9759       77.7683
# gbytes                    5.25372      0.328358      0.328205      0.330112
#
#                          [time]       [calls]        <%mpi>      <%wall>
# MPI_Bcast                 1.6727         32000         61.03         12.50
# MPI_Recv                 0.725848          4015         26.48          5.43
# MPI_Send                 0.320127          4015         11.68          2.39
# MPI_Barrier             0.0223054            16          0.81          0.17
# MPI_Comm_rank          2.97837e-06            16          0.00          0.00
# MPI_Comm_size          1.43796e-06            16          0.00          0.00
############################################################################
```

IPM: Integrated Performance Monitoring

1469389

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

Powered by IPM

command: /work/01125/yye00/IPM/Benchmark/Defiant.exe perturb -runperturbed -da_grid_x 50 -da_grid_y 50 -da_grid_z 50 -seed_phi 3454345 -seed_k11 56756756 -seed_k22 235759 -seed_k33 234656 -seed_flowmask 3222111 -percentage_phi 0.1 -percentage_k11 0.1 -percentage_k22 0.1 -percentage_k33 0.1 -percentage_flowmask 0.15 -endtime 1.0 -ksp_type bicg -pc_type bjacobi

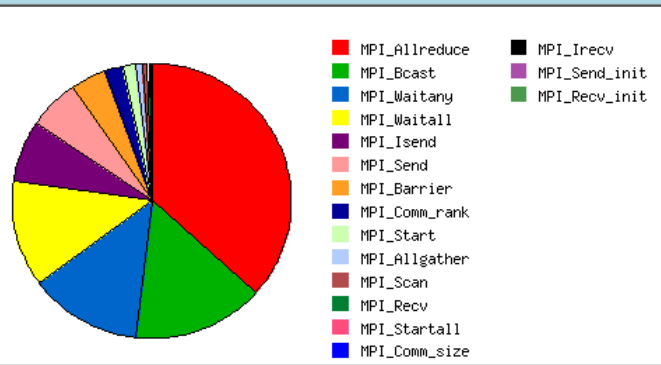
codename:	unknown	state:	running
username:	yye00	group:	G-801077
host:	i115-108 (x86_64_Linux)	mpi_tasks:	64 on 4 hosts
start:	07/13/10/00:28:10	wallclock:	3.80580e+00 sec
stop:	07/13/10/00:28:13	%comm:	11.5752798360397
total memory:	10.85705 gbytes	total gflop/sec:	1.02192900310053
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop
PAPI_FP_OPS	3889256866	*
PAPI_TOT_CYC	93055641837	*
PAPI_TOT_INS	82058705179	*
PAPI_VEC_INS	8293137711	*

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)
PAPI_FP_OPS	*	60769638.53	53254674 (63)	68822066 (21)
PAPI_TOT_CYC	*	1453994403.70	1346848050 (23)	1646491906 (12)
PAPI_TOT_INS	*	1282167268.42	1134309464 (0)	1477795580 (12)
PAPI_VEC_INS	*	129580276.73	113002002 (63)	146915653 (21)

TACC

IPM: Event Statistics

Communication Event Statistics (100.00% detail, 9.9012e-06 error)							
	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Allreduce	8	79680	4.178	8.225e-06	8.882e-04	14.82	1.72
MPI_Bcast	4	1024	4.047	5.914e-08	6.413e-02	14.35	1.66
MPI_Allreduce	512	39936	3.803	1.660e-05	1.170e-01	13.49	1.56
MPI_Allreduce	4	25472	2.250	6.012e-07	1.552e-02	7.98	0.92
MPI_Barrier	0	64	1.176	1.814e-02	1.865e-02	4.17	0.48
MPI_Isend	8	630	1.028	3.427e-07	1.647e-02	3.65	0.42
MPI_Isend	4	4556	0.943	2.738e-07	1.833e-02	3.34	0.39
MPI_Send	14976	144	0.722	1.030e-03	7.308e-03	2.56	0.30
MPI_Comm_rank	0	106948	0.620	3.725e-08	9.872e-03	2.20	0.25
MPI_Waitany	0	6093	0.542	4.615e-07	1.358e-02	1.92	0.22
MPI_Waitany	1248	27462	0.519	5.183e-07	2.723e-04	1.84	0.21
MPI_Send	16224	144	0.517	4.283e-04	7.129e-03	1.83	0.21
MPI_Waitany	1352	20370	0.496	5.197e-07	5.783e-03	1.76	0.20
MPI_Start	0	269196	0.396	3.623e-07	3.685e-05	1.40	0.16
MPI_Send	13824	48	0.298	4.035e-03	7.227e-03	1.06	0.12
MPI_Waitany	1152	10980	0.243	5.383e-07	2.310e-04	0.86	0.10
MPI_Bcast	216	576	0.231	2.302e-06	5.843e-03	0.82	0.09
MPI_Allgather	4	9088	0.215	5.118e-07	1.793e-03	0.76	0.09
MPI_Waitall	184	11	0.210	1.633e-02	2.135e-02	0.74	0.09
MPI_Scan	4	384	0.144	2.259e-05	1.600e-03	0.51	0.06
MPI_Waitany	147	453	0.141	4.866e-07	1.406e-02	0.50	0.06
MPI_Waitany	4	448	0.132	4.345e-07	5.805e-03	0.47	0.05
MPI_Waitall	320	18	0.120	3.002e-06	1.284e-02	0.42	0.05
MPI_Send	17576	42	0.108	6.682e-05	6.406e-03	0.38	0.04
MPI_Waitall	72	6	0.103	1.547e-02	2.038e-02	0.36	0.04
MPI_Waitall	96	38	0.091	2.882e-06	1.563e-02	0.32	0.04
MPI_Waitany	624	140	0.088	1.126e-06	7.880e-03	0.31	0.04
MPI_Recv	8	9	0.085	8.373e-07	7.696e-02	0.30	0.03

IPM: Integrated Performance Monitoring

When to use IPM?

- To quickly find out where your code is spending most of its time (in both computation and communication)
- For performing scaling studies (both strong and weak)
- When you suspect you have load imbalance and want to verify it quickly
- For a quick look at the communication pattern
- To find out how much memory you are using per task
- To find the relative communication & compute time

IPM: Integrated Performance Monitoring

When IPM is **NOT** the answer

- When you already know where the performance issues are
- When you need detailed performance information on exact lines of code
- When want to find specific information such as cache misses

Advanced Profiling Tools
: the next level

Advanced Profiling Tools

Can be intimidating:

- Difficult to install
- Many dependences
- Require kernel patches

} Not your problem

Useful for serial and parallel programs

Extensive profiling and scalability information

Analyze code using:

- Timers
- Hardware registers (PAPI)
- Function wrappers

PAPI

PAPI is a **P**erformance **A**pplication **P**rogramming **I**nterface

icl.cs.utk.edu/papi

API to use hardware counters

Behind Tau, HPCToolkit

Multiplatform:

- Most Intel & AMD chips
- IBM POWER 4/5/6
- Cray X/XD/XT
- Sun UltraSparc I/II/III
- MIPS
- SiCortex
- Cell

Available as a module on Lonestar and Stampede

HPCTOOLKIT

HPCToolkit

hpctoolkit.org

Binary-level measurement and analysis – no recompilation!

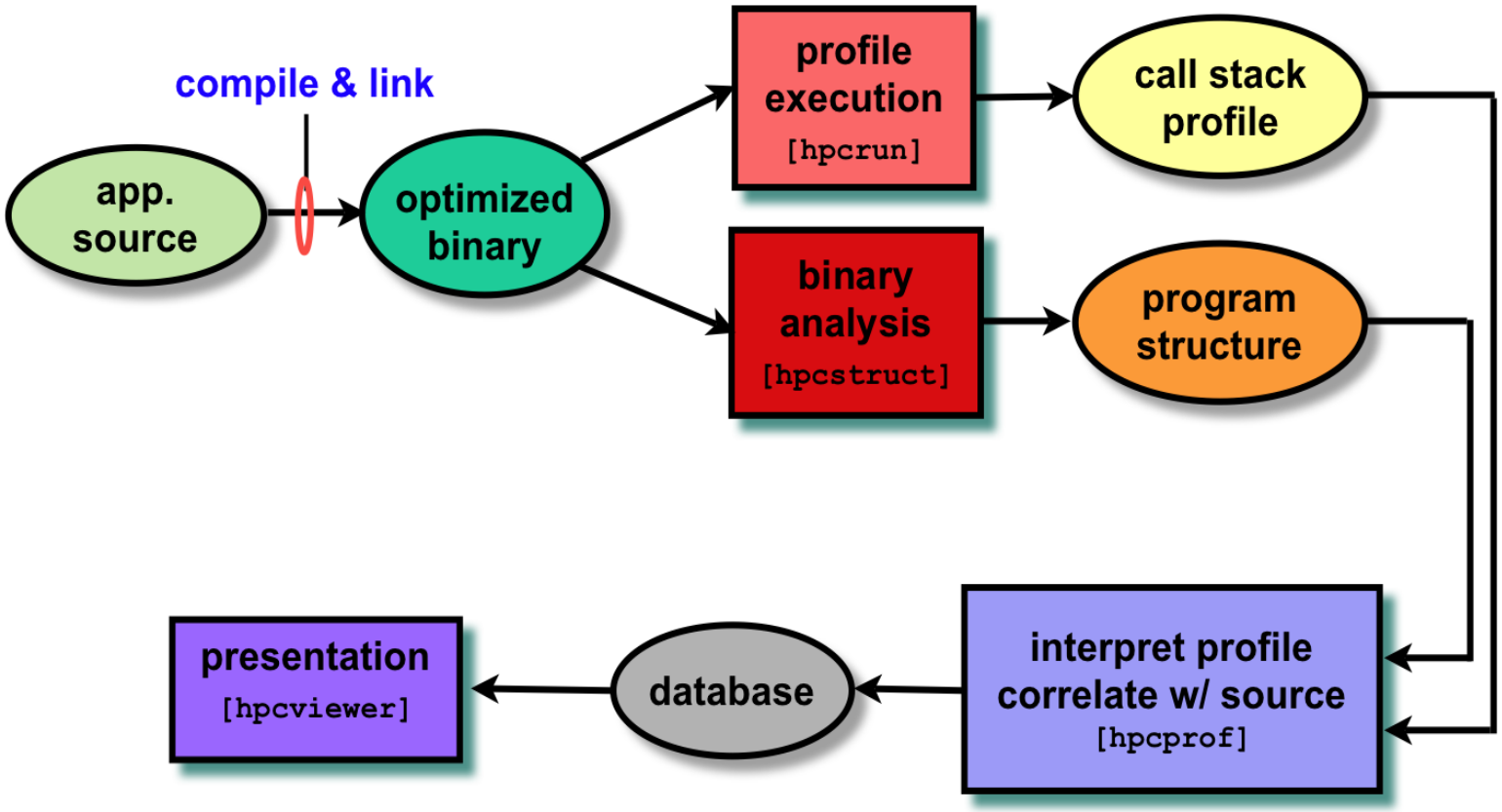
Uses sample-based measurements

- Controlled overhead
- Good scalability for large scale parallelism analysis

Good analysis tools

- Ability to compare multiple runs at different scales
- Scalable interface for trace viewer
- Good derived metrics support

HPCToolkit: Overview



HPCToolkit: Overview

Compile with symbols and optimization:

icc -g -O3 mysource -o myexe

Collect profiling information:

(Serial / OMP) **hpcrun** myexe

(MPI parallel) ibrun **hpcrun** myexe

Perform static binary analysis of the executable:

hpcstruct myexe

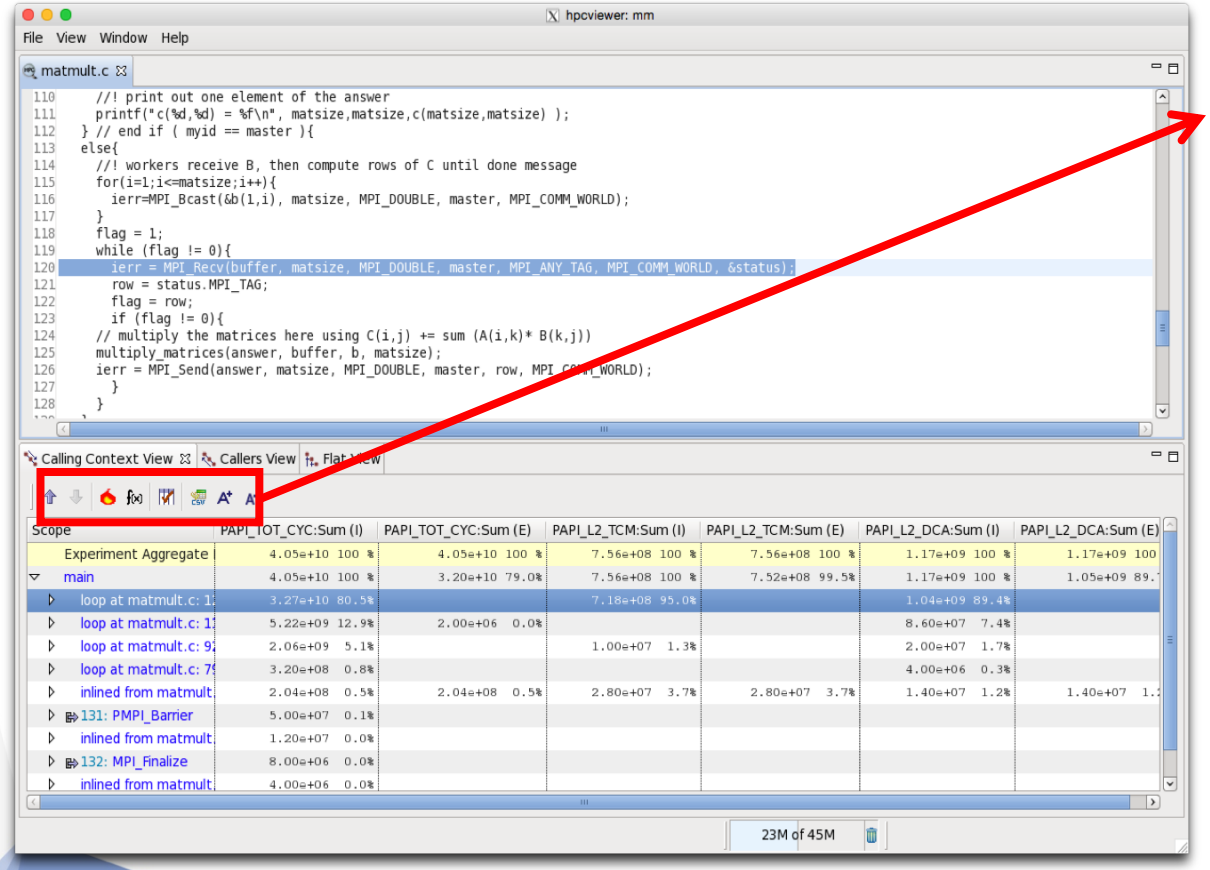
Interpret profile and correlate it with source code:

hpcprof -S myexe.hpcstruct hpctoolkit-myexe-measurement-dir

View the results

hpcviewer hpctoolkit-database

HPCToolkit: Looking at Measurements



Toolbar with useful shortcuts:

- Hot Path
- Derived Metric
- Hide/Show columns
- Export data to CSV
- Change font size

TAU

Tau

TAU is a suite of **T**uning and **A**nalysis **U**tilities

www.cs.uoregon.edu/research/tau

10+ year project involving

- University of Oregon Performance Research Lab
- LANL Advanced Computing Laboratory
- Research Centre Julich at ZAM, Germany

Integrated toolkit

- Performance instrumentation
- Measurement
- Analysis
- Visualization

Tau: Measurements

- Parallel profiling
  - Function-level, block (loop)-level, statement-level
  - Supports user-defined events
  - TAU parallel profile data stored during execution
  - Hardware counter values (multiple counters)
  - Support for callgraph and callpath profiling

- Tracing
  - All profile-level events
  - Inter-process communication events
  - Trace merging and format conversion

Tau: Results

You can also enable tracing by using:

```
export TAU_TRACE=1
```

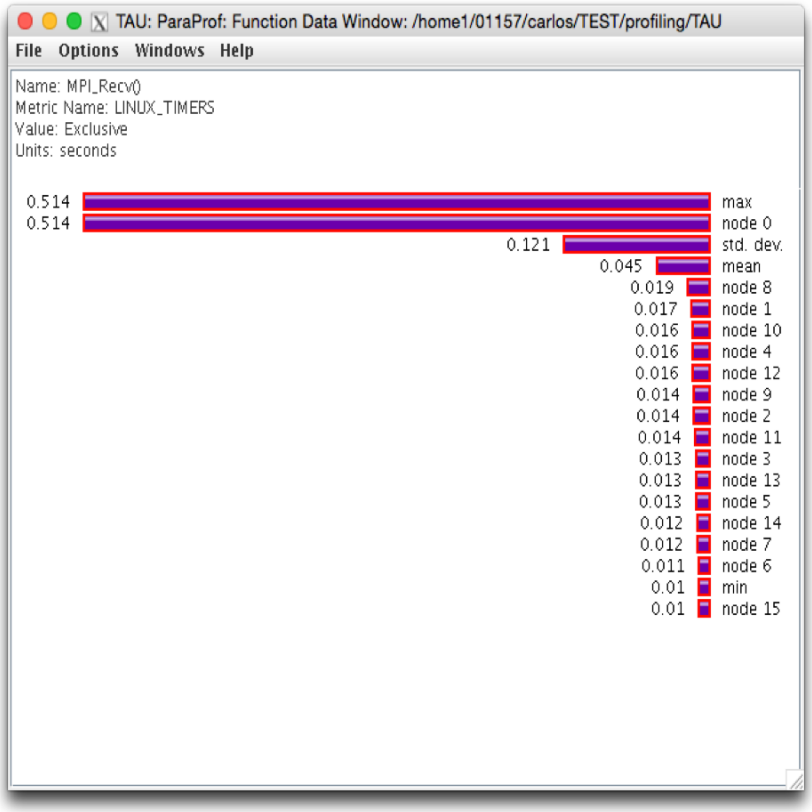
Run code normally:

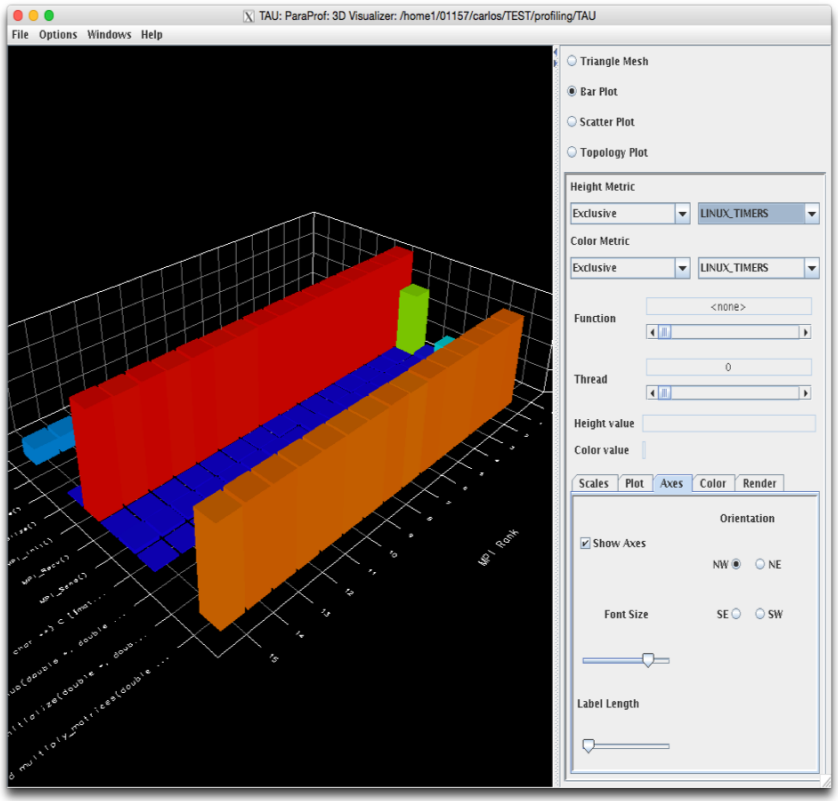
```
ibrun myexe
```

Then look at the results using paraprof:

```
paraprof /path/to/results
```

Tau: Function Data Window and 3D Plots





VTUNE AMPLIFIER

VTune

This is an Intel product

software.intel.com/en-us/intel-vtune-amplifier-xe

Available on stampede: `module load vtune`

Supports serial, threaded, MPI, and hybrid applications

Two part interface: command line and gui

- Sort, filter, and visualize results on the timeline and on your source

Helps to correlate timing information with original source code and assembly

Much improved support for KNL, including memory access details

- Memory bandwidth to DDR4 and MCDRAM (Flat or Cache)
- Fine grained control of memory allocation using libmemkind

Why VTune

Insight into performance, threading, bandwidth, caching and more

Analysis is simple and convenient (Good GUI)

- Sort, filter, and visualize results on the timeline and on your source

Particularly useful to look at memory access on KNL

- Memory bandwidth to DDR4 and MCDRAM (Flat or Cache)

Also useful for fine grained control of memory allocation using libmemkind

VTune Basics

VTune is easy to use by employing pre-defined **collections**

Collections have **knobs** that allow you to capture additional information

To find out which collections are available :

```
$ amplxe-cl -help collect
```

To find out which knobs are available for a collection :

```
$ amplxe-cl -help collect memory-access
```

Pre-defined Collections

Collection	Information Provided
hotspots	Identify most time-consuming code sections (user mode)
advanced-hotspots	Adds CPI, higher frequency low overhead sampling
concurrency	CPU utilization, threading synchronization overhead (user mode)
disk-io	Disk IO preview, not working in Stampede (requires root)
memory-access	Memory access details and memory bandwidth utilization. May include specific arrays that are heavy BW users (useful for fine-grained MCDRAM use)
hpc-performance	Performance characterization, including floating point unit and memory bandwidth utilization

Useful knobs & options

Option	Description
-data-limit=0	Override default maximum data collection size
-no-summary	Do not produce text summary
-no-auto-finalize	Do not finalize data analysis after collection
-finalize	Carry out data analysis after collection
-start-paused	Start application without profiling
-resume-after=X	Resume profiling after X seconds
-duration=Y	Profile only for Y seconds

knob	Description
analyze-memory-objects=true	Determine arryays using most memory bandwidth (highest L2 miss rates)
analyze-openmp=true	Determine inefficiencies in OpenMP regions

VTune: Collecting Performance Data

Result finalization and viewing on KNL target might be slow. Use the recommended workflow:

Compile with debug symbols

```
$ mpicc -g -O3 -xMIC-AVX512 -qopenmp ./code.c
```

Run collection on KNL deferring finalization on host

```
$ amplxe-cl -c hotspots -no-auto-finalize -r <result_dir> ./app
```

Finalize the result on the host

```
$ amplxe-cl -finalize -r <result_dir> -search-dir <bin_dir>
```

Generate reports, work with GUI

```
$ amplxe-cl -report hotspots -r <result_dir>
```

```
$ amplxe-gui
```

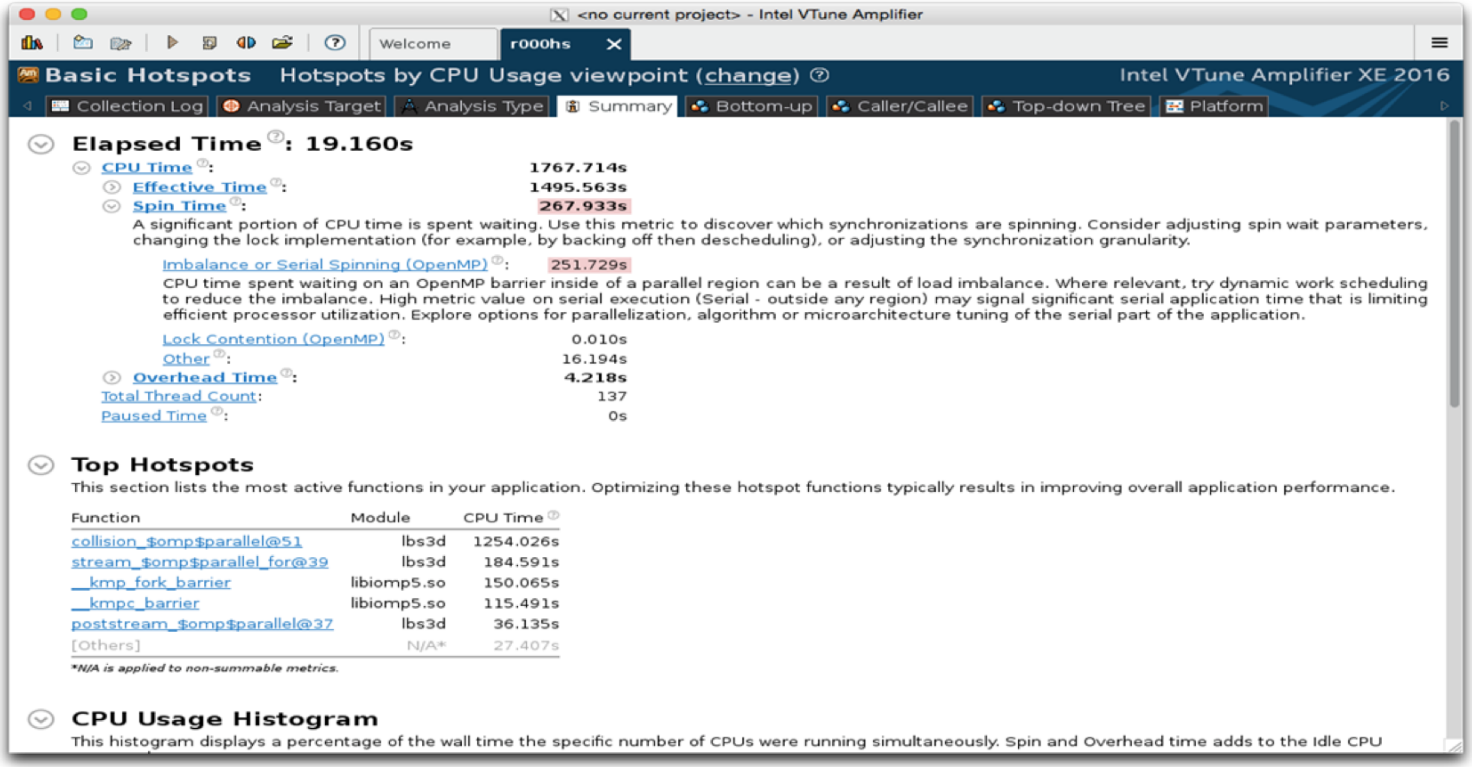
VTune in a batch Job

```
#!/bin/bash
#SBATCH -J knl-run.%j.err          # job name
#SBATCH -o knl-run.%j.out          # out/err file name
#SBATCH -N 1                       # Number of nodes
#SBATCH -n 1                       # MPI Tasks
#SBATCH -p normal                  # queue (partition)
#SBATCH -t 00:30:00                # runtime (hh:mm:ss)
#SBATCH -A <Allocation>

export OMP_NUM_THREADS=68
# Launch amplxe-cl with proper options
amplxe-cl -c hotspots -no-auto-finalize -- ./a.out
```

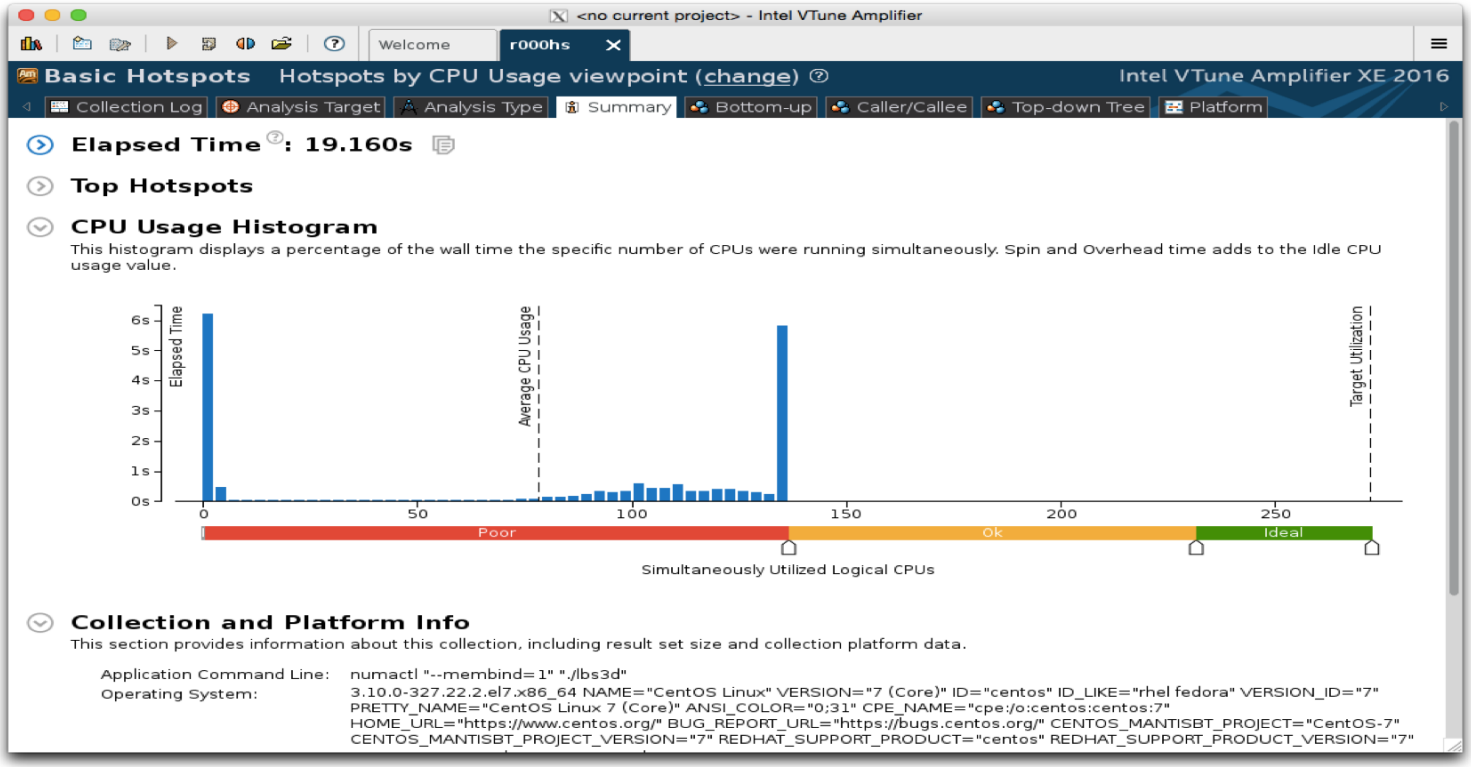
VTune: Sample Summary

```
Collection and Platform Info
----------------------------
Parameter                 hotspots.0
------------------------  ------------------------------------------------------
Application Command Line  ./mm
Operating System          2.6.32-431.17.1.el6.x86_64 CentOS release 6.5 (Final)
MPI Process Rank          0
Computer Name             c401-604.stampede.tacc.utexas.edu
Result Size               2012293
Collection start time     04:01:56 12/11/2014 UTC
Collection stop time      04:01:58 12/11/2014 UTC

CPU
---
Parameter          hotspots.0
-----------------  ----------------------------
Name               Intel(R) Xeon(R) E5 processor
Frequency          2699995693
Logical CPU Count  16

Summary
-------
Elapsed Time:       1.160
CPU Time:           0.860
Average CPU Usage:  0.714
```

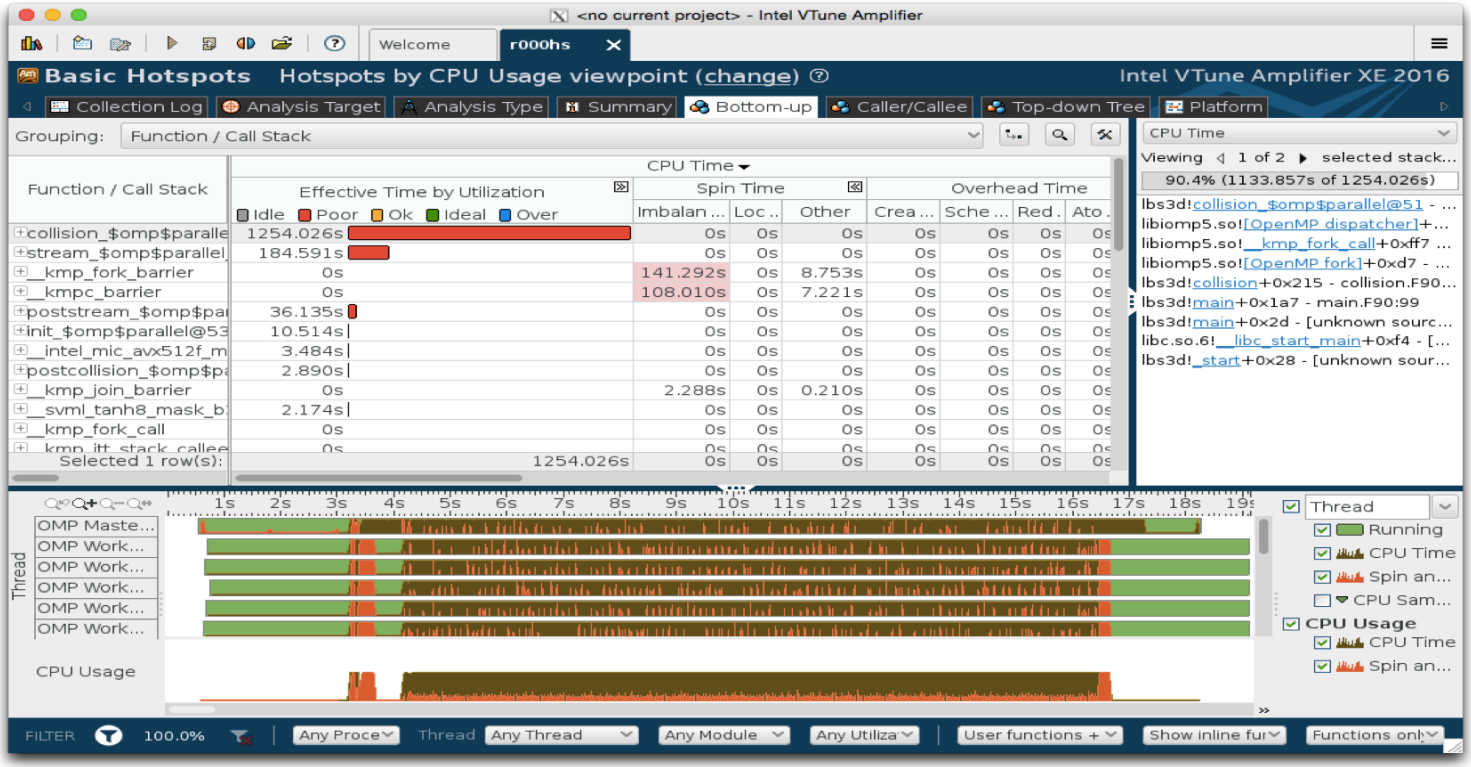
Example hotspots summary



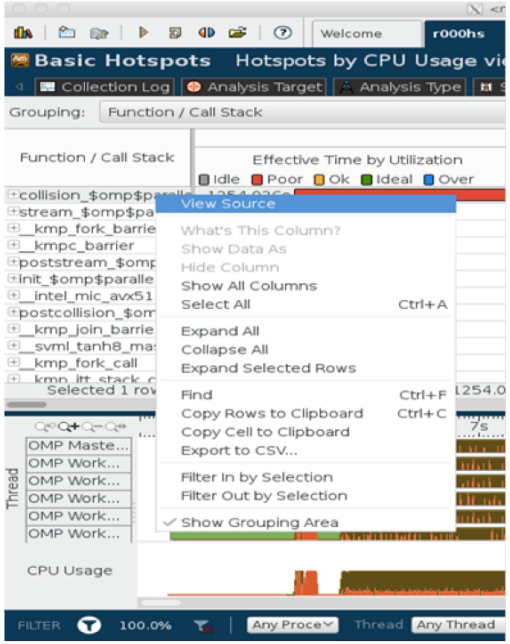
CPU histogram

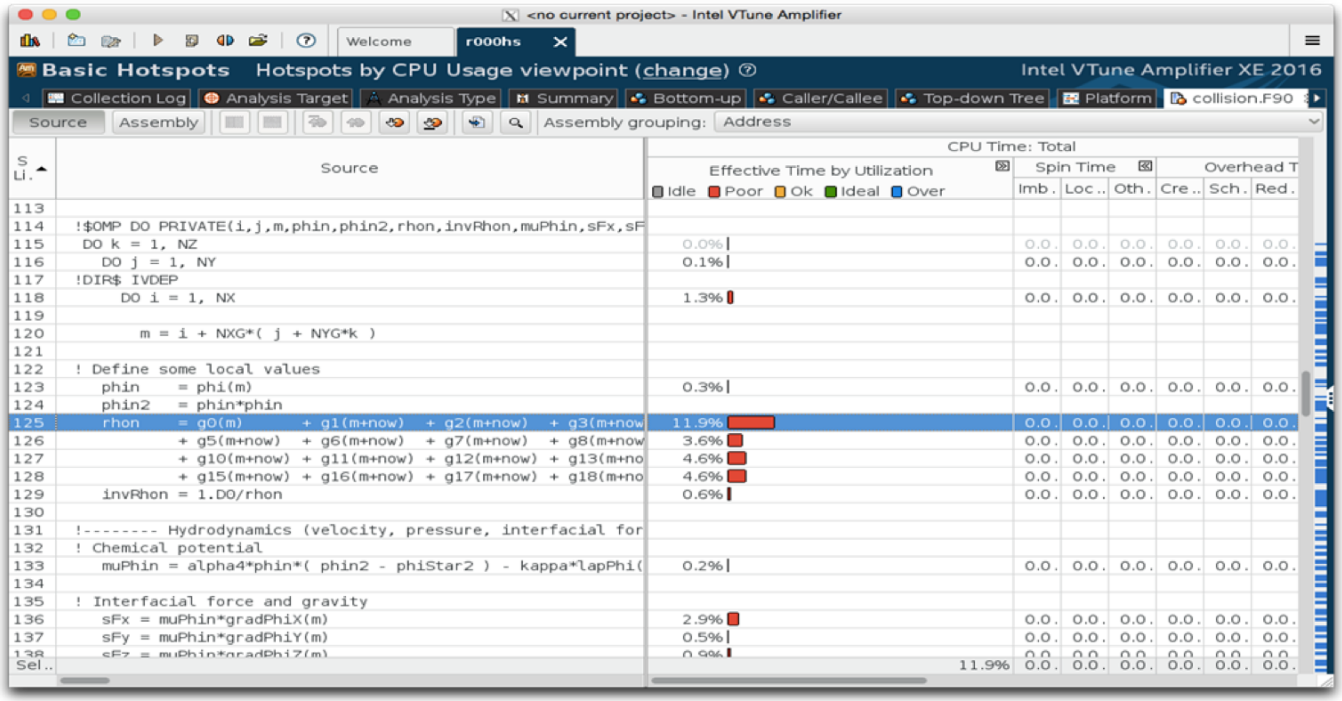


Hotspots Bottom-up View

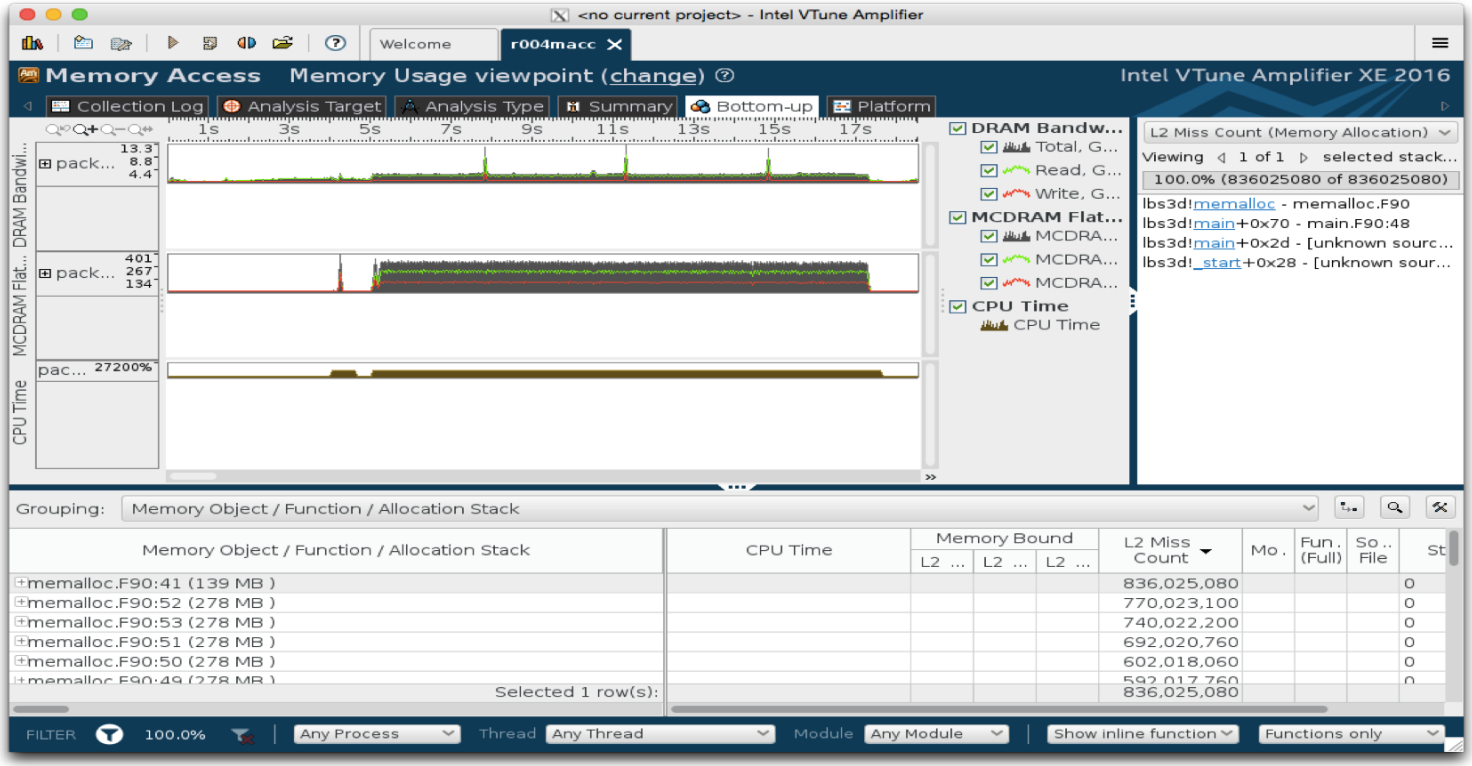


Getting to the Source

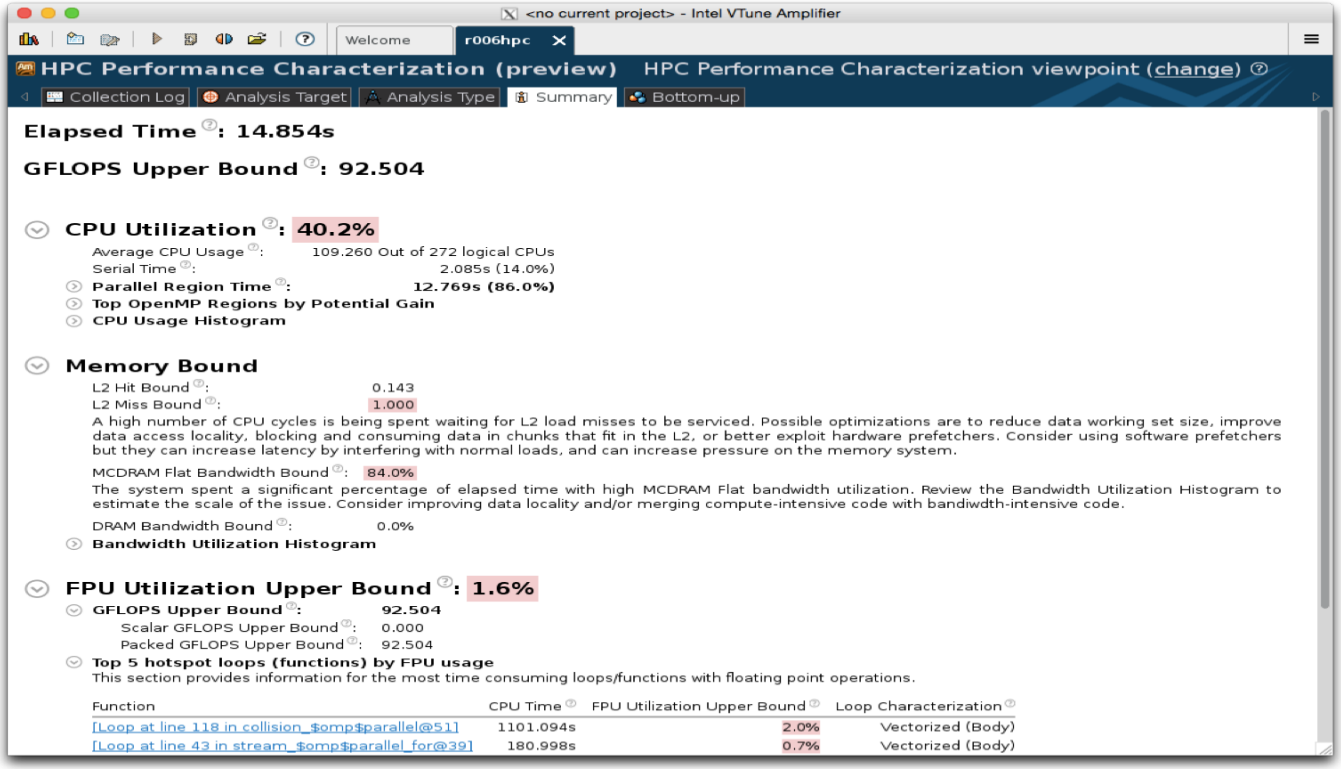




Memory-access Objects Details



hpc-performance Summary



Intel Advisor

About Advisor

- Intel Advisor is a vectorization optimization and shared memory threading assistance tool for C, C++, C# and Fortran code
- Advisor supports both serial, threaded, and MPI applications

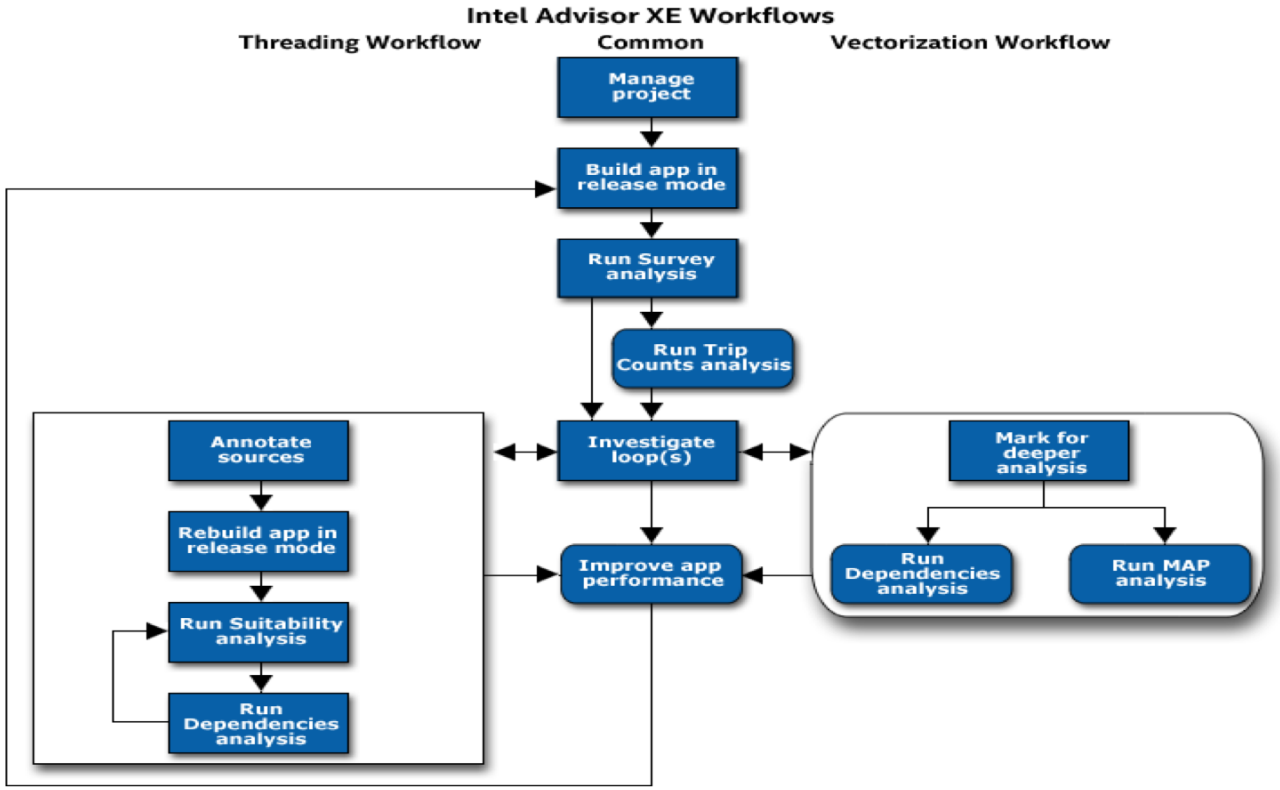
Advisor Capabilities

- Evaluate the efficiency of vectorized code
- Discover where to add parallelism to your code by identifying where your code spends its time
  - You propose parallel code regions when you annotate the parallel sites and tasks.
- Predict the performance you might achieve with the proposed parallel code regions.
- Predict the data sharing problems that could occur in the proposed parallel code regions

Tool	Description	Target Program Requirements
Survey	Helps you discover and select the best places to add parallelism in your program.	Moderate optimization. Limited function inlining.
Trip Counts	Helps you to collect loop iteration statistics.	Moderate optimization. Run survey analysis first
Suitability	Helps you predict the likely performance impact of adding parallelism to the selected places.	Moderate optimization. Limited function inlining.
Dependencies	Helps you predict and eliminate data sharing problems before you add parallelism. 50 to several hundred times slower.	Optimization disabled. **Minimize execution time.**
Memory Access Patterns (MAP)	Helps you to collect data on memory access strides. 50 to several hundred times slower.	Optimization disabled. **Minimize execution time.**

Intel Advisor Reference: https://software.intel.com/en-us/node/527404

How to use Advisor: workflows



Intel Advisor Reference:
https://software.intel.com/en-us/node/608339

Using Advisor

Setup environment

```
$source /opt/intel/advisor_2018.1.0.523188/advixe-vars.sh
```

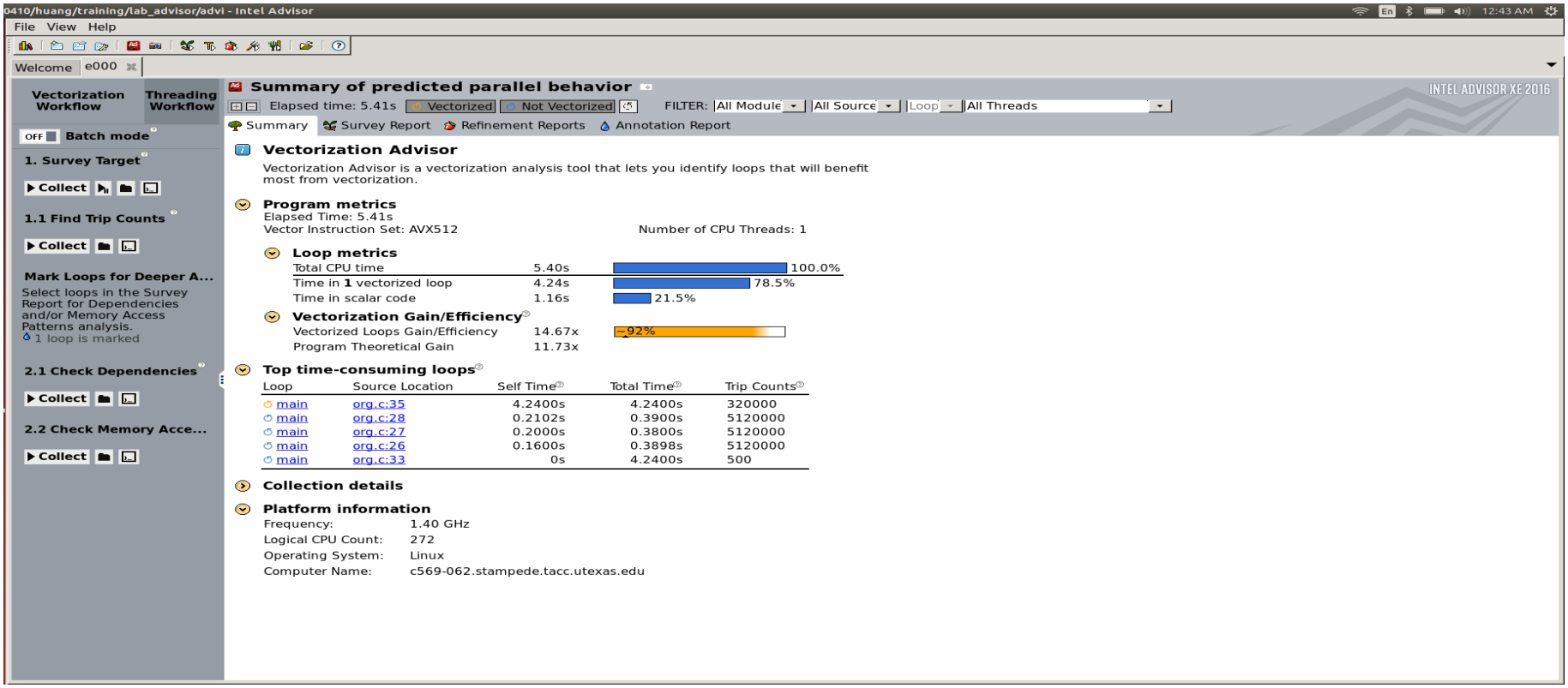
Compile with optimization and debug symbols

```
$ icc -g -xMIC-AVX512 -O2 -qopt-report=5 demo.c
```

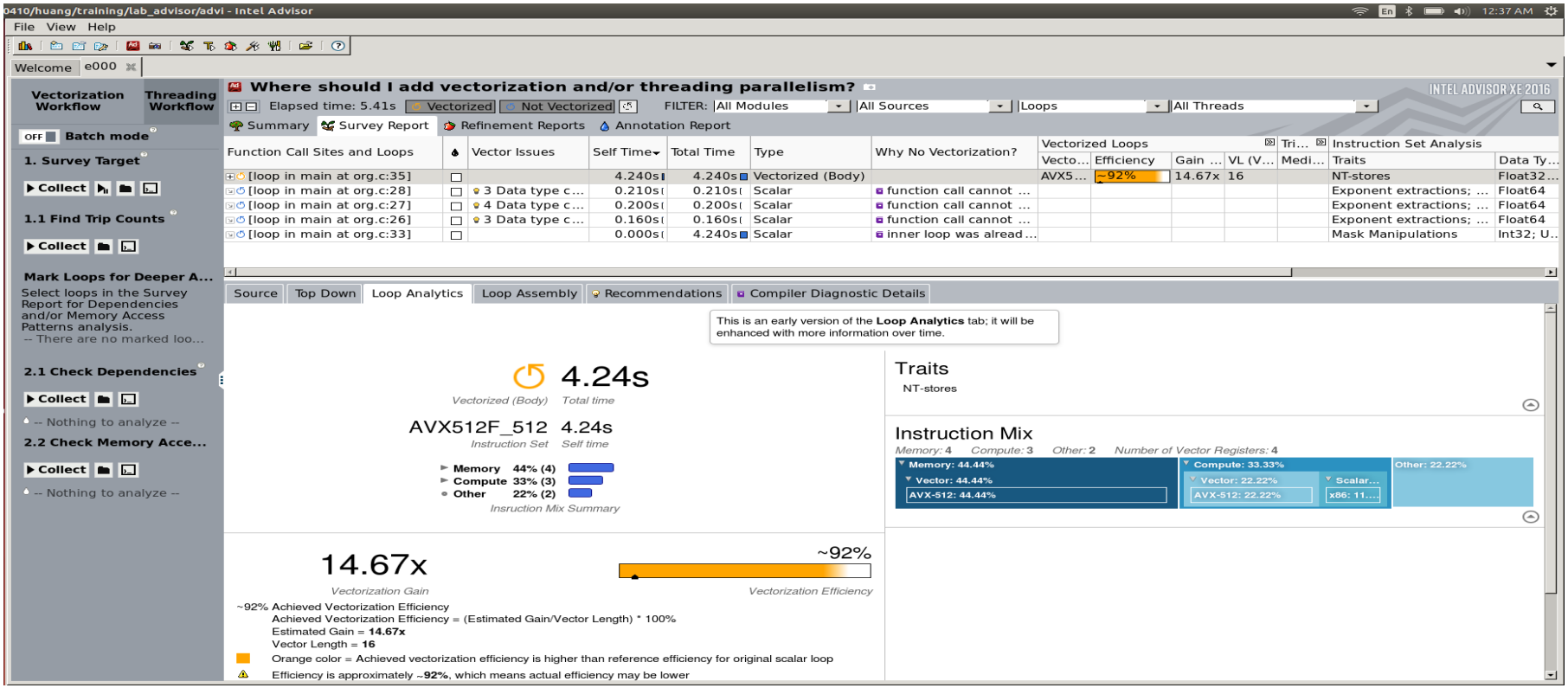
Collect data

```
$ advixe-cl -c survey -- ./a.out
```

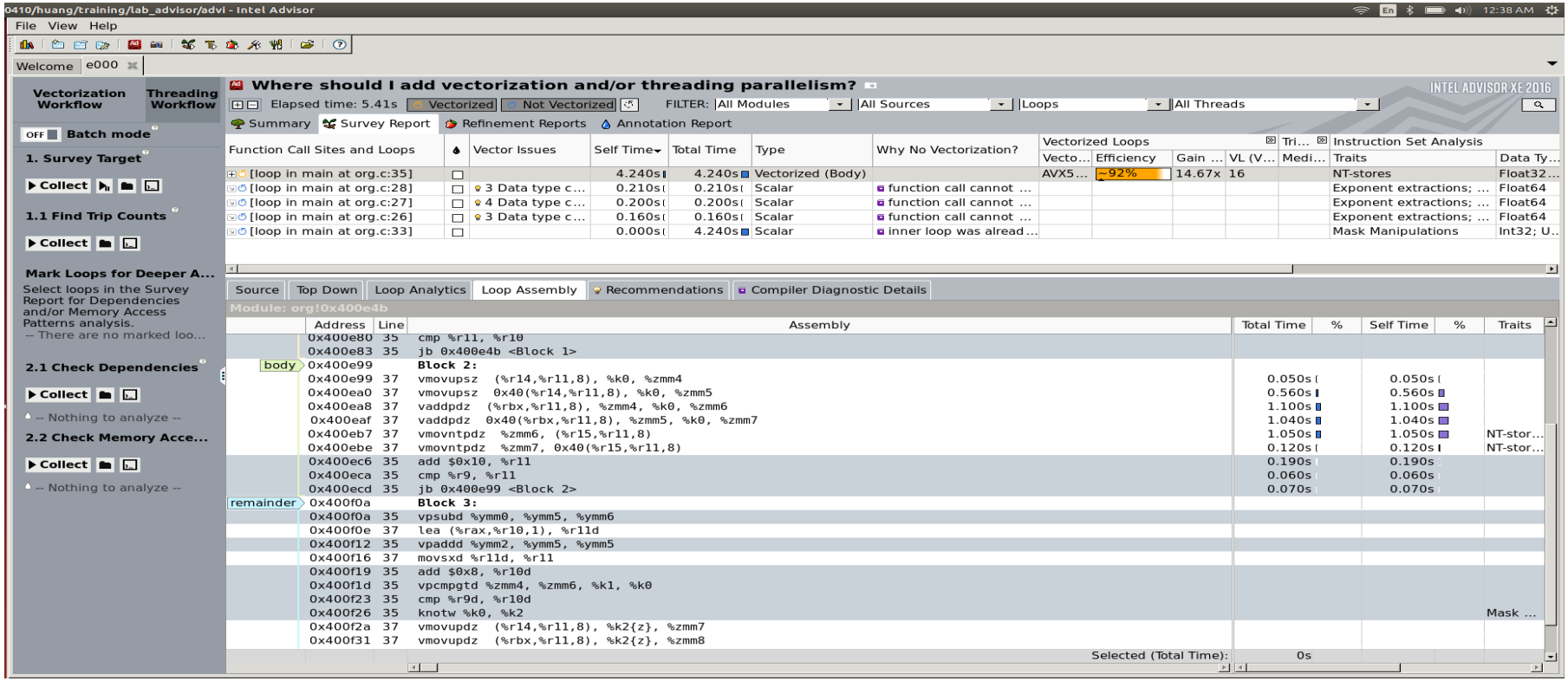
Summary


0410/huang/training/lab_advisor/advi - Intel Advisor
File View Help
Welcome e000
Vectorization Workflow
Threading Workflow
OFF Batch mode
1. Survey Target
Collect
1.1 Find Trip Counts
Collect
Mark Loops for Deeper A...
Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.
1 loop is marked
2.1 Check Dependencies
Collect
2.2 Check Memory Acce...
Collect
Summary of predicted parallel behavior
INTEL ADVISOR XE 2016
Elapsed time: 5.41s
Vectorized
Not Vectorized
FILTER: All Module All Source Loop All Threads
Summary
Survey Report
Refinement Reports
Annotation Report
Vectorization Advisor
Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.
Program metrics
Elapsed Time: 5.41s
Vector Instruction Set: AVX512
Number of CPU Threads: 1
Loop metrics
Total CPU time 5.40s 100.0%
Time in 1 vectorized loop 4.24s 78.5%
Time in scalar code 1.16s 21.5%
Vectorization Gain/Efficiency
Vectorized Loops Gain/Efficiency 14.67x ~92%
Program Theoretical Gain 11.73x
Top time-consuming loops
Loop Source Location Self Time Total Time Trip Counts
main org.c:35 4.2400s 4.2400s 320000
main org.c:28 0.2102s 0.3900s 5120000
main org.c:27 0.2000s 0.3800s 5120000
main org.c:26 0.1600s 0.3898s 5120000
main org.c:33 0s 4.2400s 500
Collection details
Platform information
Frequency: 1.40 GHz
Logical CPU Count: 272
Operating System: Linux
Computer Name: c569-062.stampede.tacc.utexas.edu

Survey, loop analytics



Survey, loop Assembly



Profiling do and don't

DO

Test every change you make

Profile typical cases

Compile with optimization flags

Test for scalability

DO NOT

Assume a change will be an improvement

Profile atypical cases

Profile *ad infinitum*

- Set yourself a goal or
- Set yourself a time limit

Other tools

Valgrind* valgrind.org

Powerful instrumentation framework, often used for debugging memory problems

MPIP mpip.sourceforge.net

Lightweight, scalable MPI profiling tool

Scalasca www.fz-juelich.de/jsc/scalasca

Similar to Tau, complete suit of tuning and analysis tools.

MPIP https://github.com/TACC/remora

REsource MOnitoring for Remote Applications