TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

# Introduction to Manycore Tools: Intel VTune & Advisor

Todd Evans

Feb 3rd, 2017

# Intel VTune

## Line-level Performance Profiling Tool

- Supports serial, threaded, MPI, and hybrid profiling
- Shows users where runtime is spent in code
- Samples hardware counters at regular intervals and correlates values to location on code
- Specialized support for KNL including memory access details

# Why Use VTune?

- Provides insight into performance, threading performance, scalability, bandwidth, caching and more
- Analysis is simple and convenient (Good GUI)
  - Sort, filter, and visualize results on the timeline and on your source
- Useful to look at memory access on KNL
  - DDR4 vs MCDRAM
- Useful for fine grained control of memory allocation using `libmemkind` (`hbw_malloc`)

# How-to Use VTune

1. Compile your code with:
   - `-g` flag: provides symbol table information
   - Link to libmemkind `-lmemkind`: provides memory access information
   - `icc -g -O2 -xMIC-AVX512 -qopenmp -lmemkind omp_mm_knl.c`
   - `mpicc -g -O2 -xMIC-AVX512 -qopenmp -lmemkind omp_mm_knl.c`
2. Run the program with VTune Amplifier:
   - Graphical User Interface: `amplxe-gui`
   - Command Line Interface: `amplxe-cl`
3. Analyze the results with VTune Amplifier
   - Graphical User Interface: `amplxe-gui`
   - Command Line Interface: `amplxe-cl`

# Run your job with VTune `amplxe-cl`

## Job submission script

```
#!/bin/bash
#SBATCH -J knl-run.%j.err       # job name
#SBATCH -o knl-run.%j.out      # output and error file name
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p normal                 # queue (partition)
#SBATCH -t 00:30:00               # run time (hh:mm:ss)
#SBATCH -A Your_Project

export OMP_NUM_THREADS=16
#Launch amplxe-cl with proper options
amplxe-cl -collect memory-access -knob analyze-mem-objects=true \\
                -r output_dir ./a.out
```

# Run you job with VTune `amplxe-cl`

```
amplxe-cl <-action> [-action-option]
[-global-option] [[--] <target> [target-options]]
```

| | |
|---|---|
| `<-action>`: | The action to perform, such as collect or report collect, report, help, etc. |
| `[-action-option]`: | Action-options modify behavior specific to the action. |
| `[-global-option]`: | Global-options modify behavior in the same manner for all actions. For example: -q, -quiet to suppress non-essential messages. |
| `<target>`: | The target application to analyze. |
| `[target-options]`: | Options for the application. |

Complete `amplxe-cl` command syntax from Intel:

`https://software.intel.com/en-us/node/544220`

# Run your job with VTune `amplxe-cl`

## Tips

- VTune is easiest to use by employing pre-defined *collections*
- Collections have *knobs* that allow you to capture additional information
- Examples
  - `amplxe-cl -collect hotspots -r output_dir ./a.out`
  - `amplxe-cl -collect memory-access -knob analyze-mem-objects=true -r output_dir ./a.out`
  - `amplxe-cl -collect hotspots -r output_dir ibrun -np 16 ./a.out -other-options`
- For help: `amplxe-cl -help` or `amplxe-cl -help collect`

# Pre-defined Collections

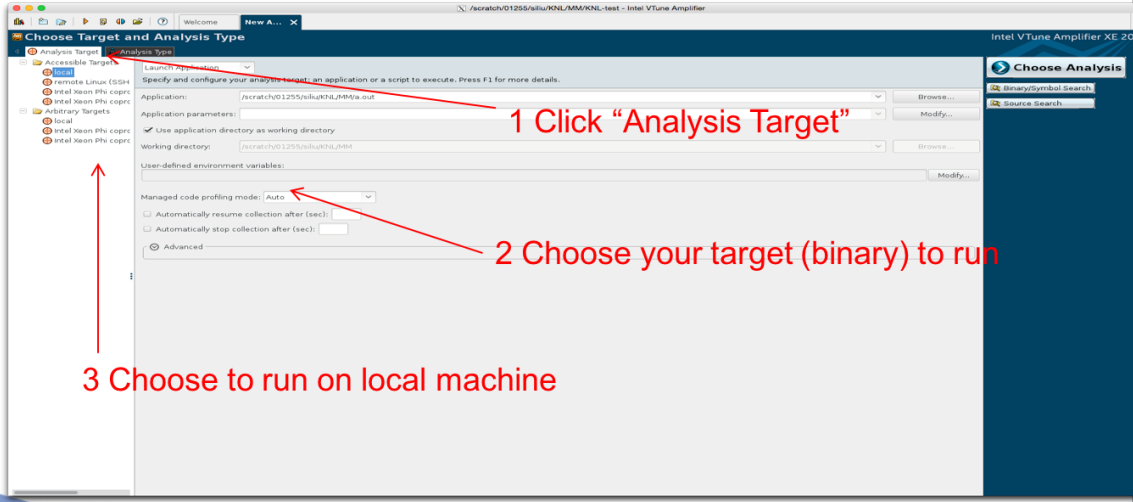
| Collection | Information Provided |
|---|---|
| hotspots | Identify most time-consuming code sections (user mode) |
| advanced-hotspots | Adds CPI, higher frequency low overhead sampling |
| concurrency | CPU utilization, threading synchronization overhead (user mode) |
| disk-io | Disk IO preview,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

# Useful knobs and options

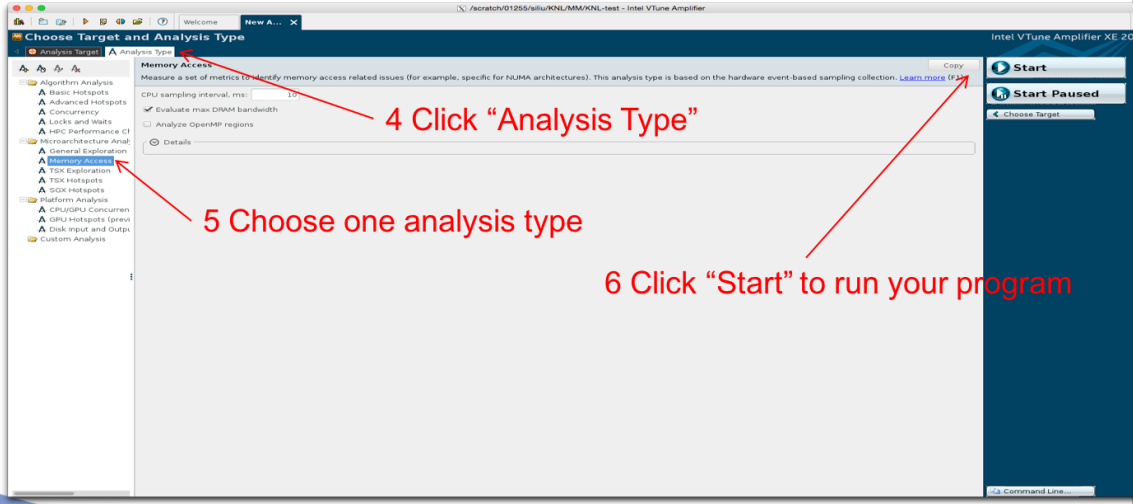
| Option | Description |
|---|---|
| -data-limit=0 | Override default maximum data collection size |
| -no-summary | Do not produce text summary |
| -no-auto-finalize | Do not finalize data analysis after collection |
| -finalize | Carry out data analysis after collection |
| -start-paused | Start application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

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

# Run your job with VTune `amplxe-gui`



# Run your job with VTune `amplxe-gui`



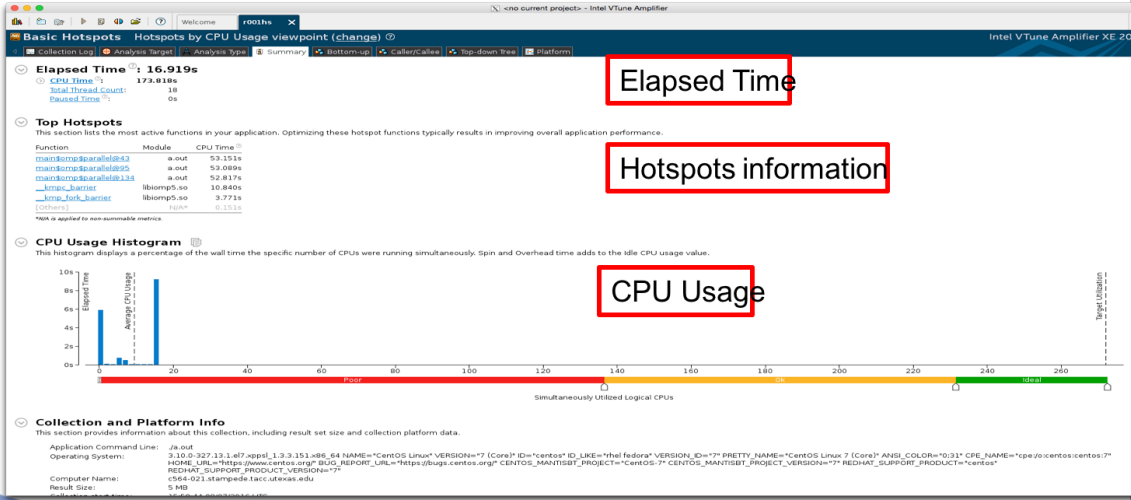
# `amplxe-cl` to analyze performance

It is possible to use the VTune command line (`amplex-cl`) to analyze performance. However, using the Graphic User Interface (`amplxe-gui`) is strongly recommended!

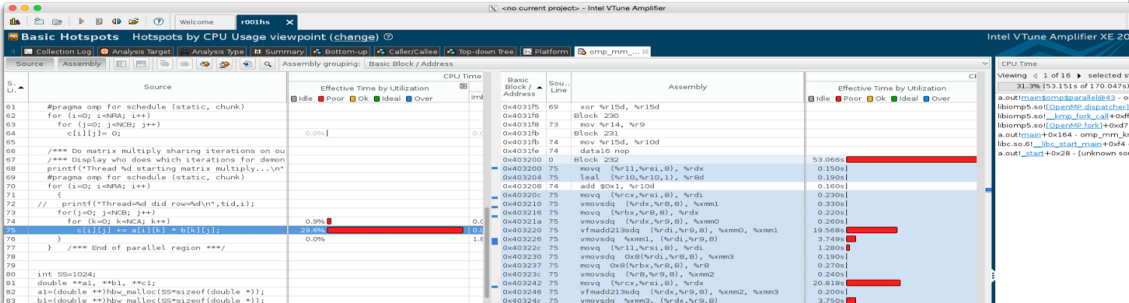
**Example**

```
amplxe-cl -report hotspots -result-dir r000hs -group-by
function
```

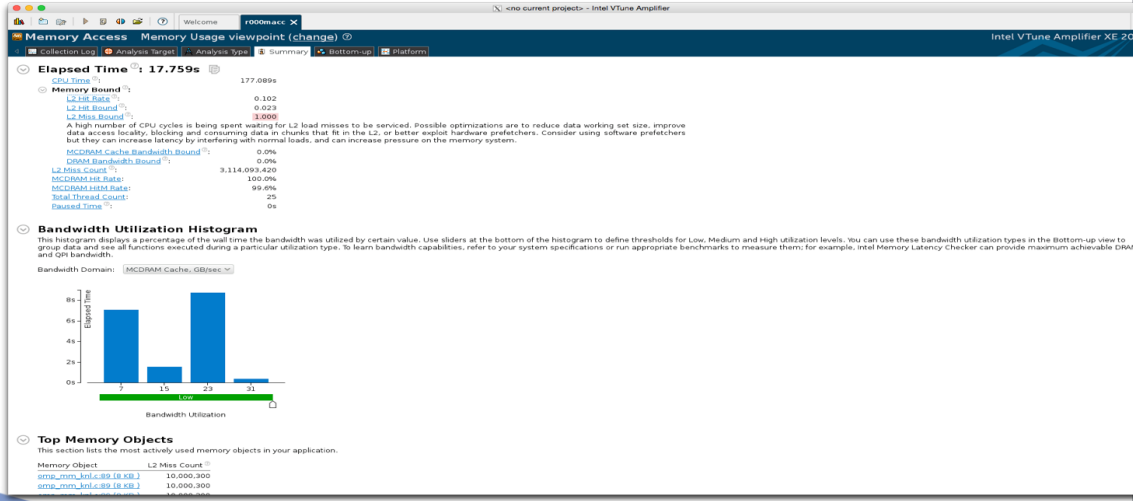
# amplxe-gui: Summary



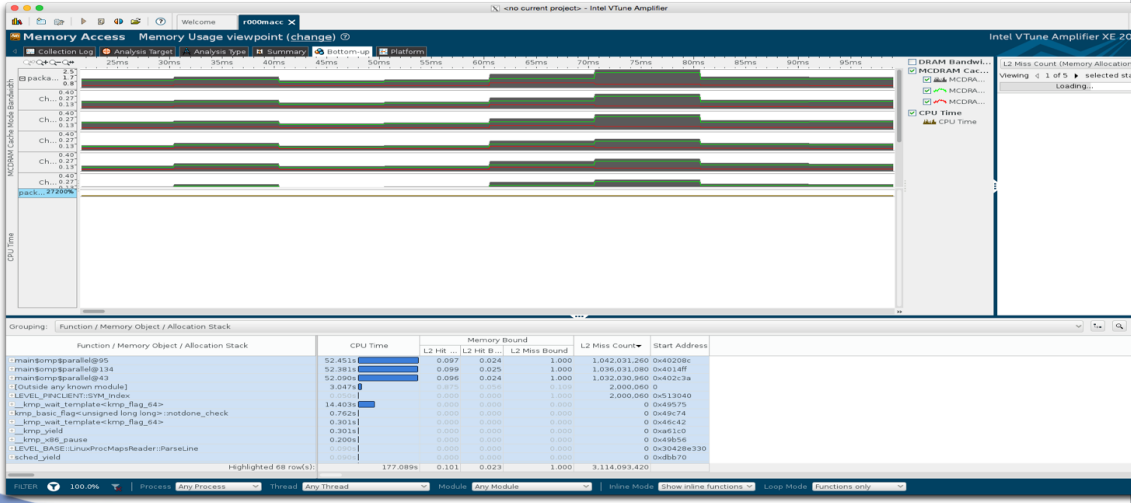
# amplxe-gui: Analyze Hot Subtrees



# `amplxe-gui`: Memory Access



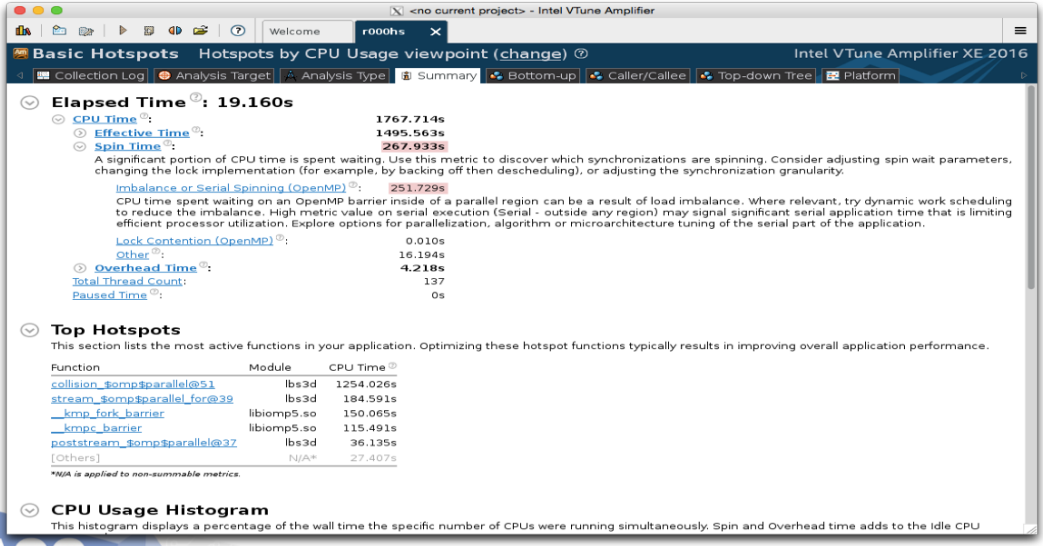
# `amplxe-gui`: Memory Access



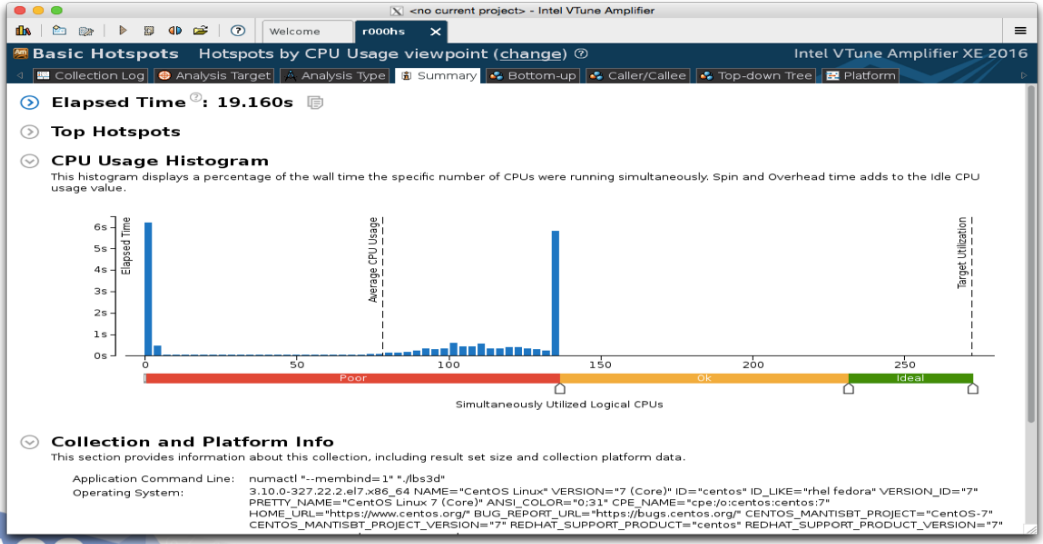
# hotspots

**getting the basics right**

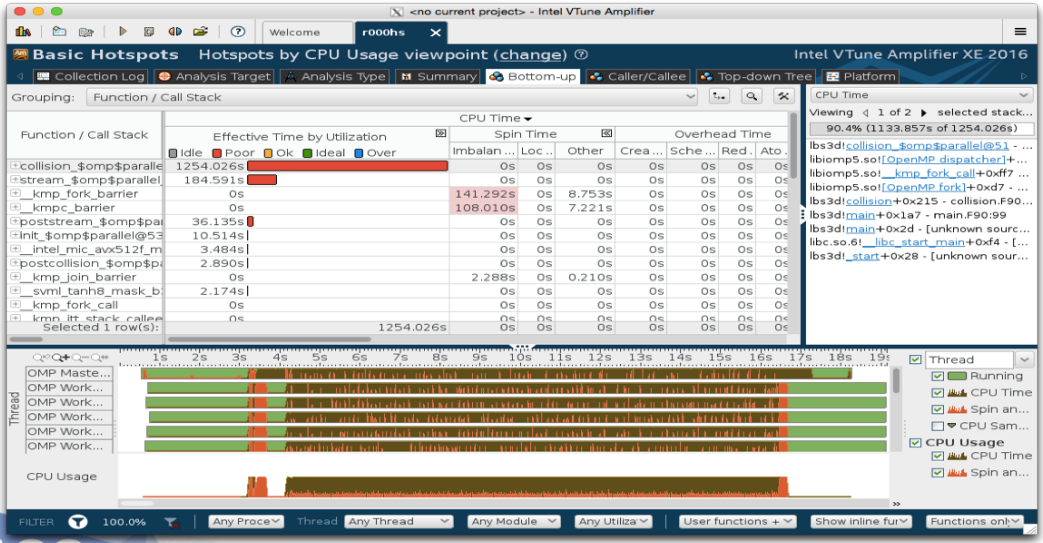
# hotspots Summary



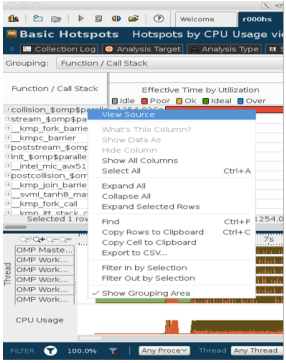
# hotspots CPU Histogram

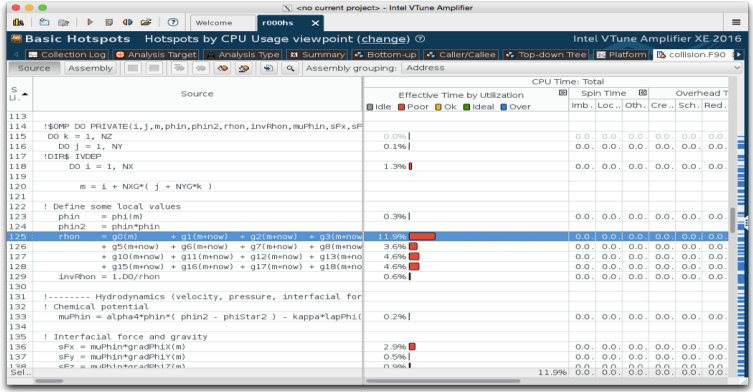


# hotspots Bottom-up View

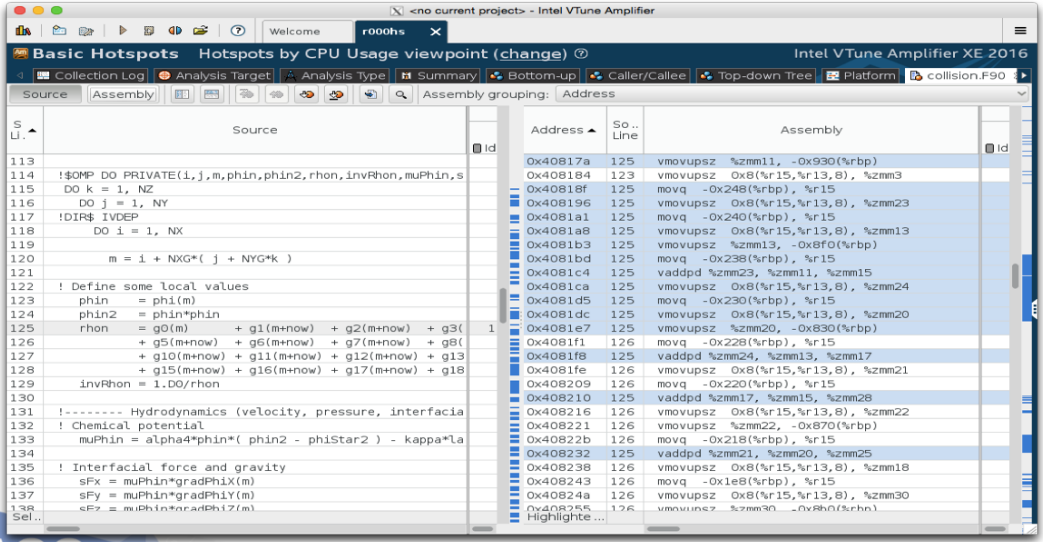


# hotspots Getting to the Source



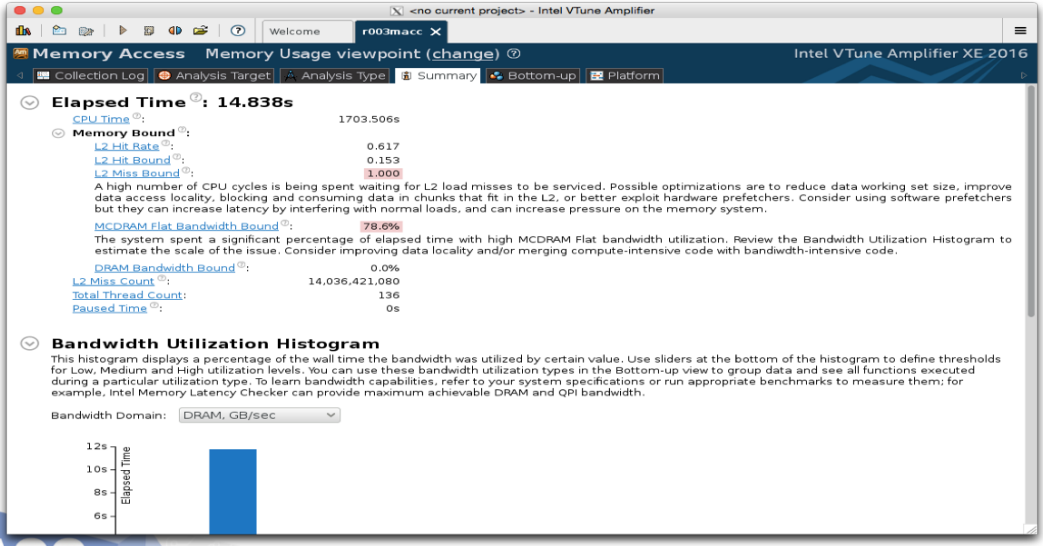
# hotspots Digging into the Assembly



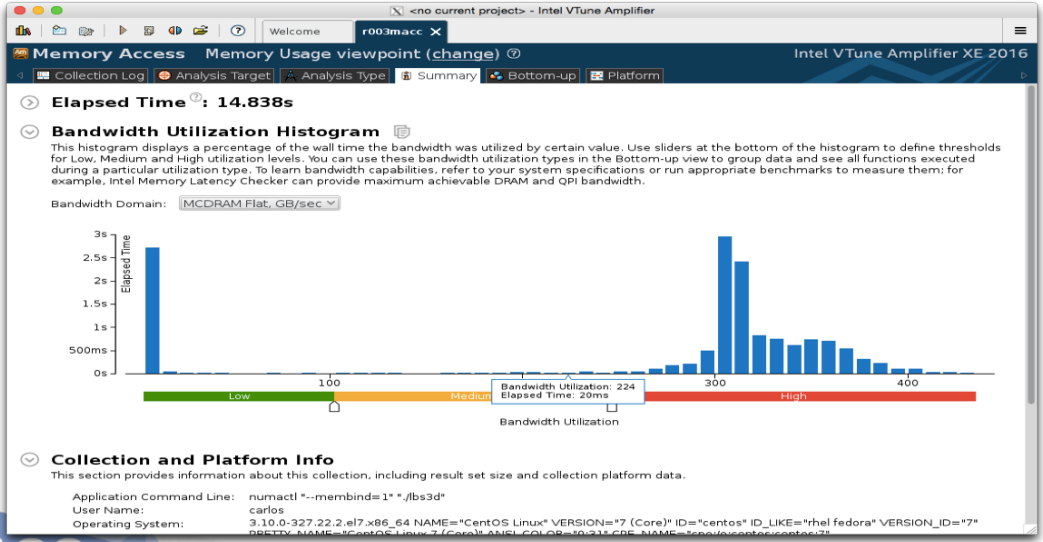
# memory-access

**It's all about data movement. . .**

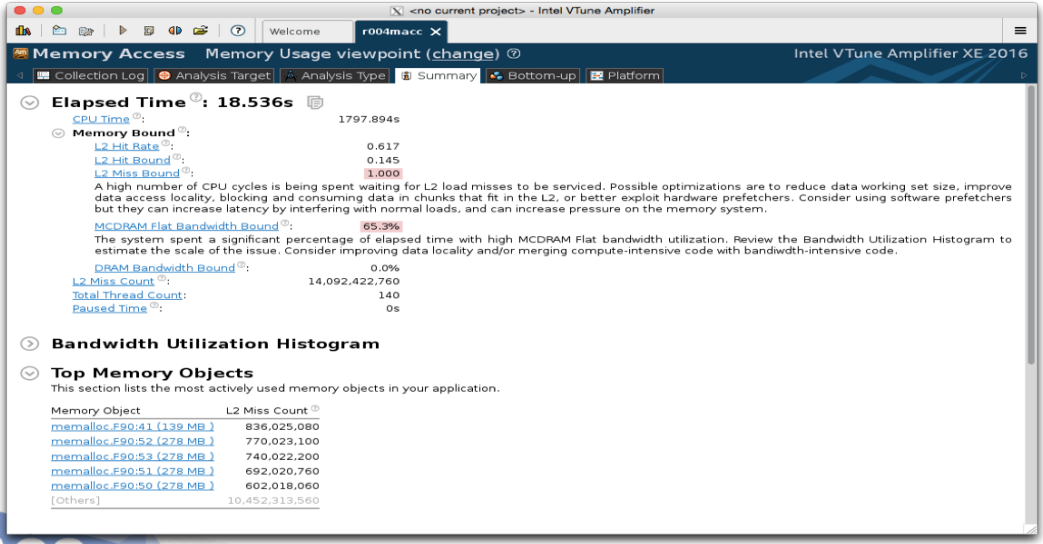
# memory-access Summary



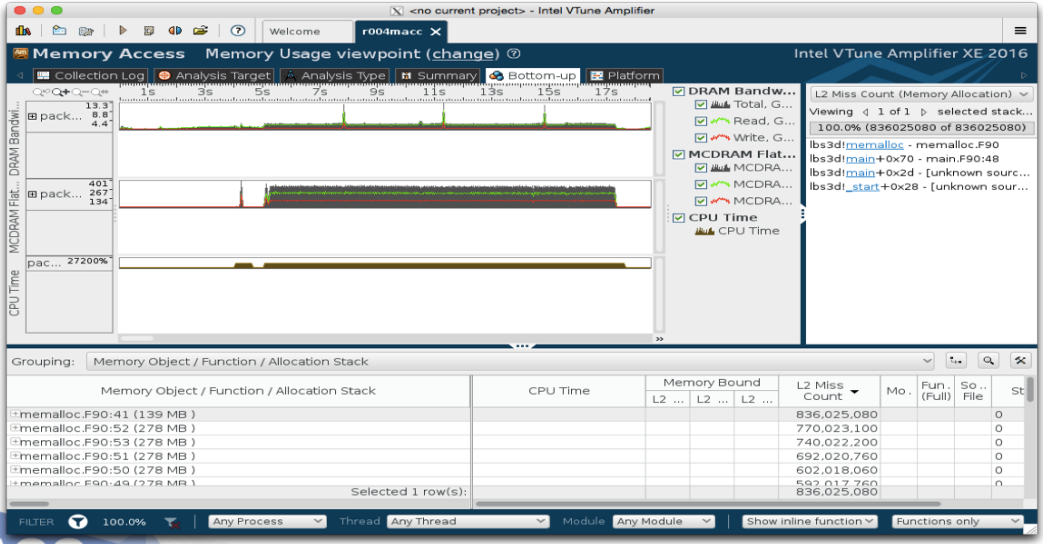
# memory-access Bandwidth Histogram



# memory-access With Objects Knob



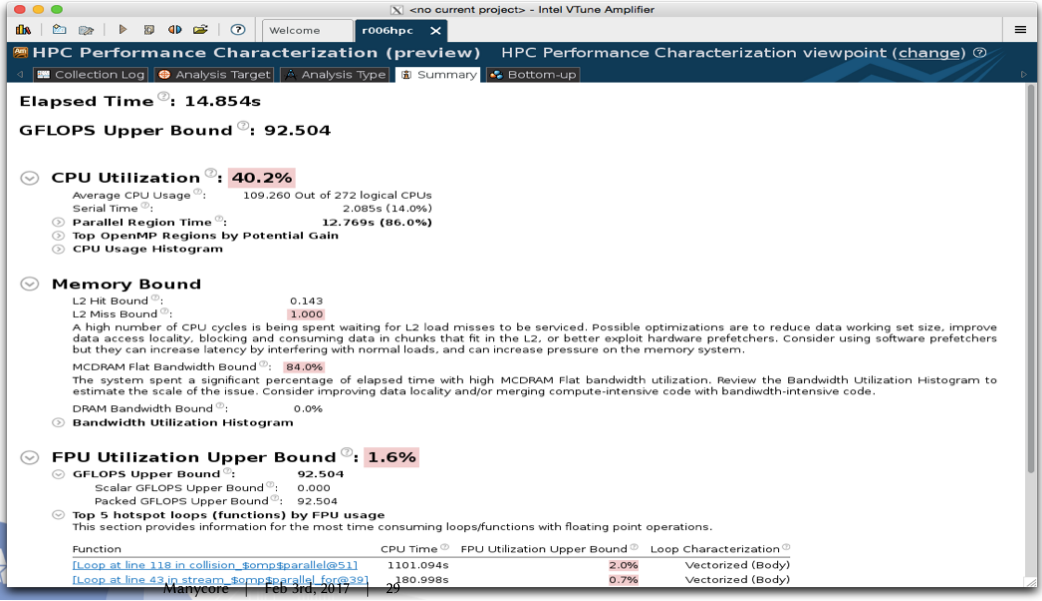
# memory-access Object Details



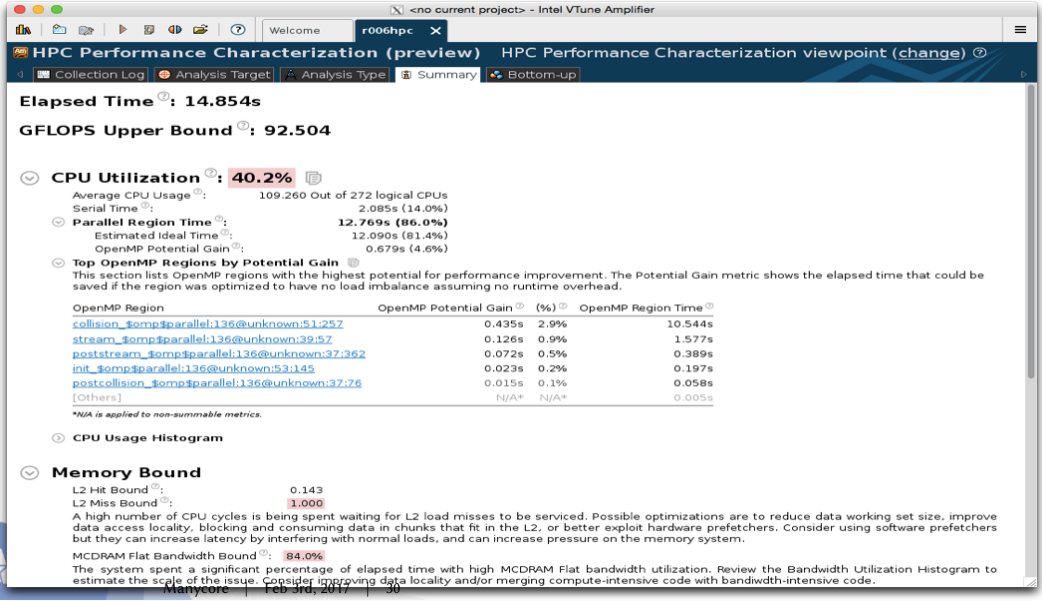
# hpc-performace

**It's all about performance. . .**

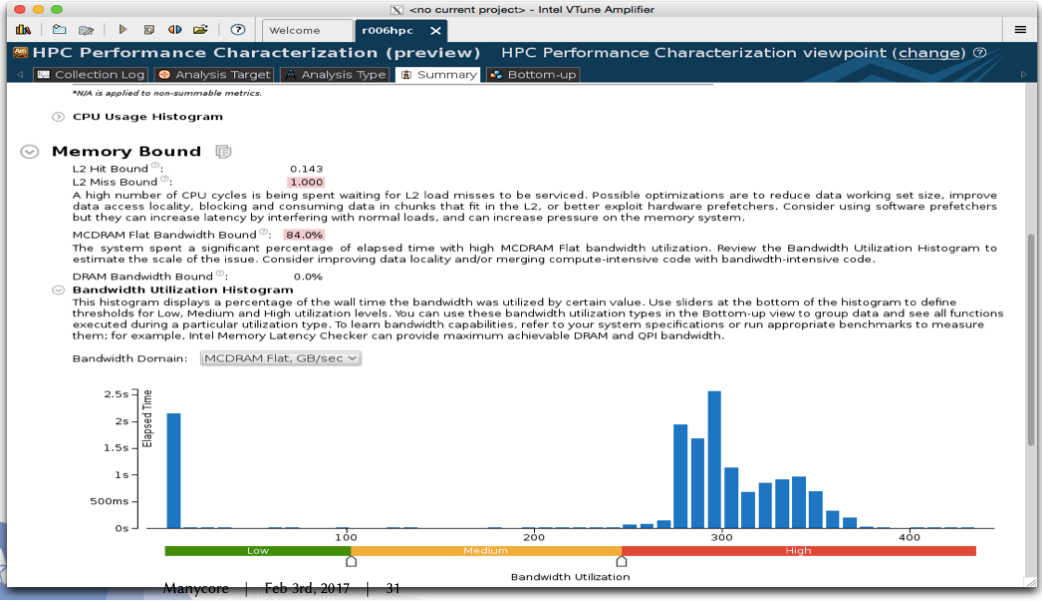
# hpc-performance Summary



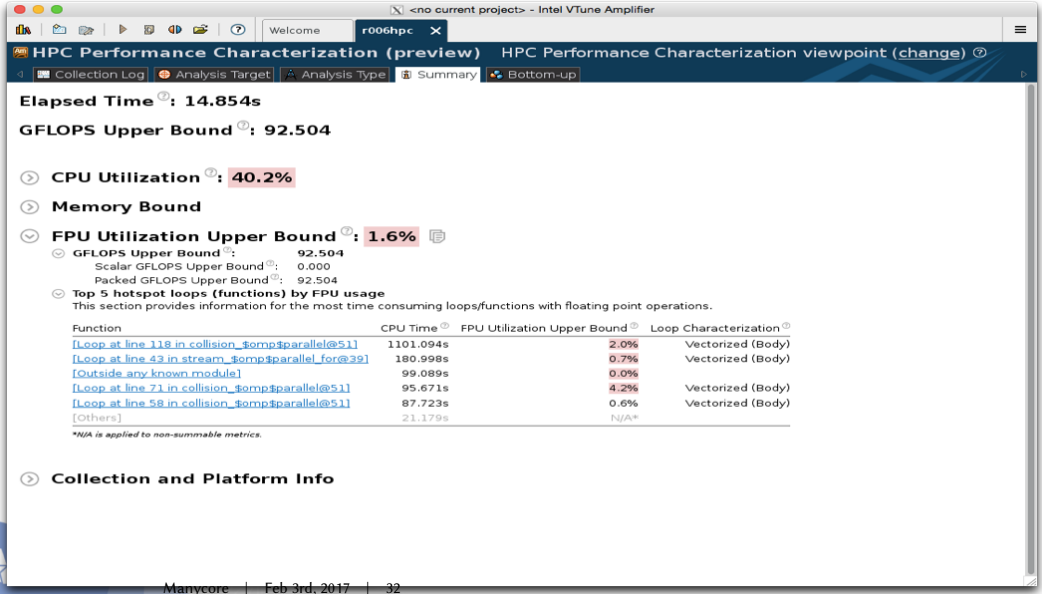
# hpc-performance OpenMP Analysis



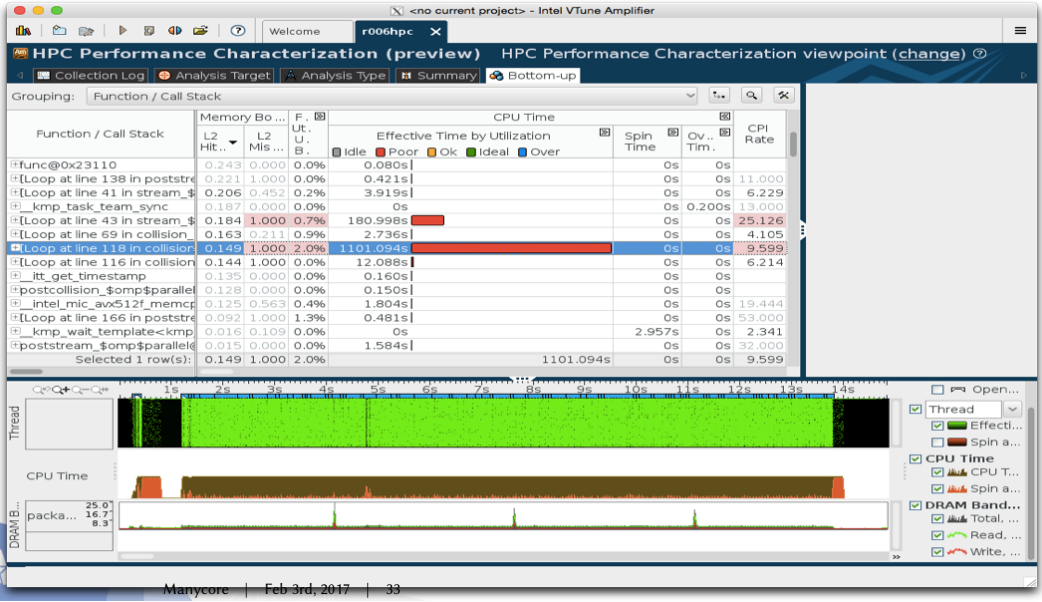
# hpc-performance Memory Information



# hpc-performance FPU Information



# hpc-performance Bottom-up View



# Basic VTune Workflow

## Result finalization and viewing on KNL might be slow

- Compile with debug symbols
  - `$ mpicc -g -O3 -xMIC-AVX512 -qopenmp ./code.c`
- Run collection on KNL
  - `$ amplxe-cl -c hotspots ./app`
- Generate reports, work with GUI
  - `$ amplxe-gui`

# 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
- Current version is 2017.1.0

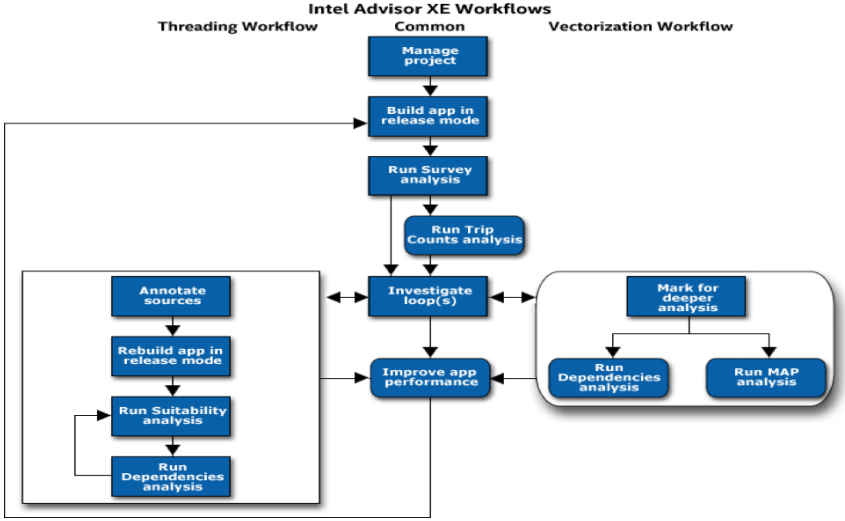
# Advisor Capabilities

- Vectorization and Roofline Advisor
  - Evaluate the efficiency of vectorized code and key SIMD bottlenecks
  - Provide precise Trip/Call Counts, FLOPS and AVX-512 mask utilization
  - Check for loop-carried dependencies dynamically
  - Identify Memory vs. Compute balance via automated Roofline and Memory Access Pattern tools
- Threading Advisor
  - 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

# Basic Advisor Tools

| Tool | Description | Target Program Requirements |
|---|---|---|
| Survey | Helps you discover and select the best places to add parallelism in your program. | Moderate optimization<br>Limited function inlining |
| Trip Counts | Helps you to collect loop iteration statistics. | Moderate optimization<br>Run survey analysis first |
| Suitability | Helps you predict the likely performance impact of adding parallelism to the selected places. | Moderate optimization<br>Limited function inlining |
| Dependencies | Helps you predict and eliminate data sharing problems before you add parallelism. 50-500 times slower. | Optimization disabled<br>**Minimize execution time** |
| Memory Access Patterns | Helps you to collect data on memory access strides. 3-20 times slower. | Optimization disabled<br>Minimize execution time |

# How-to use Advisor: Workflow



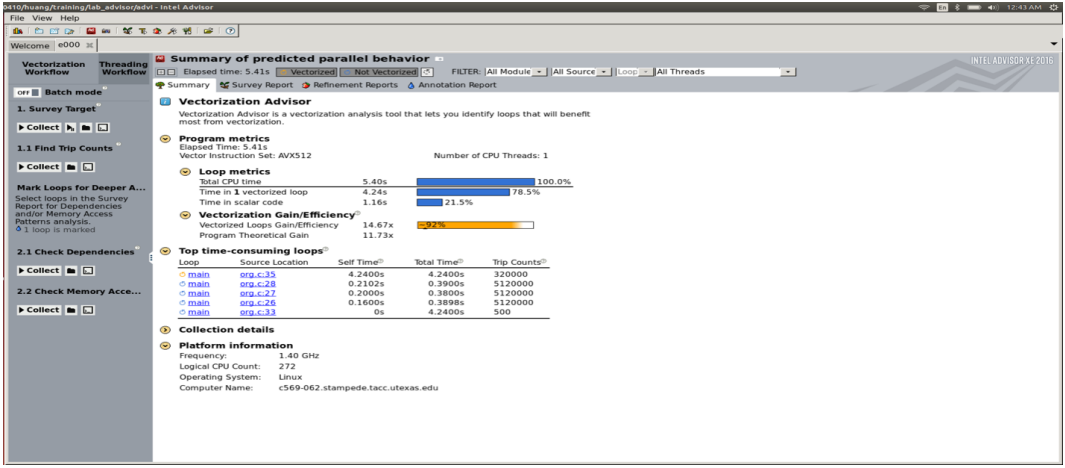

Intel Advisor Reference:

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

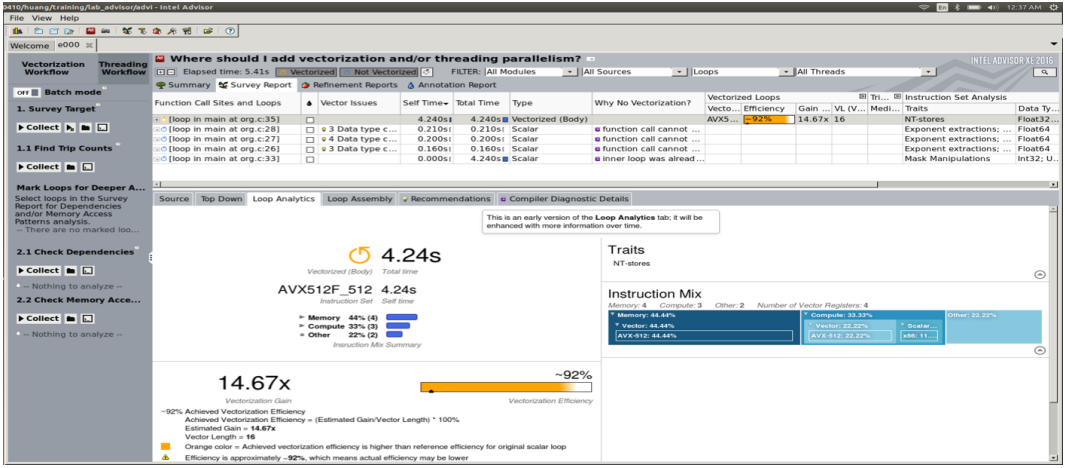
# Using Advisor

- Setup environment
  - `$ module load advisor`
  - `$ export OMP_NUM_THREADS=1` #for your OpenMP code
- Compile with optimization and debug symbols
  - `$ icc -g -xMIC-AVX512 -O2 -qopt-report=5 demo.c`
- Collect data
  - `$ advixe-cl -c survey --search-dir src:=./ ./a.out`
- Analyze the data:
  - `$ advixe-gui`

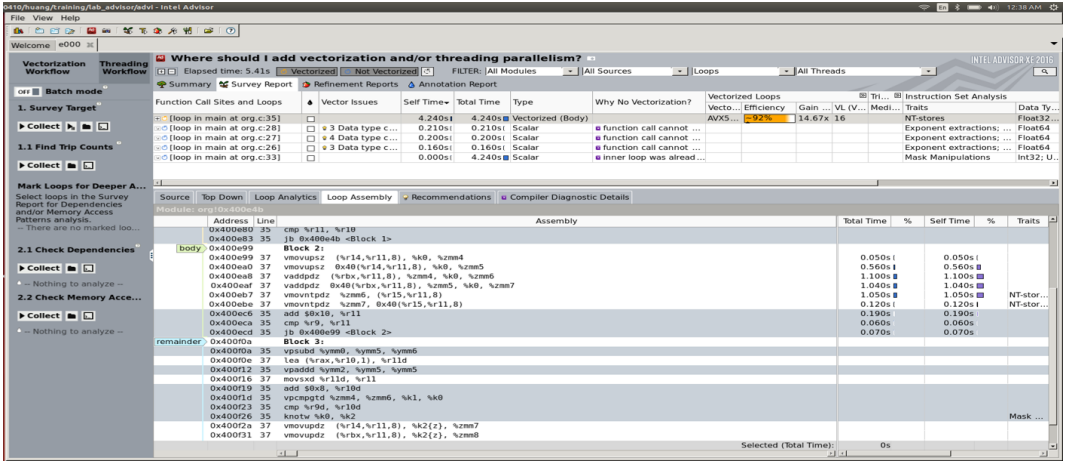
# Advisor Summary



# Survey: loop analytics



# Survey: loop assembly



# Example of multi-run analysis

- Survey
  - `$ advixe-cl -c survey --search-dir src:=./ ./a.out`
- Tripcounts
  - `$ advixe-cl -c tripcounts --search-dir src:=./ ./a.out`
- Suitability
  - Add site annotations in source code, recompile, and run suitability analysis
    ```
    $ icc -g -xMIC-AVX512 -O2 -qopt-report=5 demo.c
    -I$ADVISOR_2017_DIR/include
    ```
  - `$ advixe-cl -c suitability --search-dir src:=./ ./a.out`

# Annotating Source Code

## C/C++ Applications

```
#include <advisor-annotate.h>
...
ANNOTATE_SITE_BEGIN();
...
ANNOTATE_ITERATION_TASK (task1);
...
ANNOTATE_SITE_END();
```

## Fortan Applications

```
use advisor_annotate
...
CALL ANNOTATE_SITE_BEGIN()
...
CALL ANNOTATE_ITERATION_TASK(task1)
...
CALL ANNOTATE_SITE_END()
```

# From serial to annotated to OMP code

```
// matrix multiply routine
void multiply_d(int Size, double **aMatrix,double **bMatrix,
                                          double **product) {
  for(int i=0;i<Size;i++) {
      for(int j=0;j<Size;j++) {
        double sum = 0;
        for(int k=0;k<Size;k++) {
          sum += aMatrix[i][k] * bMatrix[k][j];
        }
        product[i][j] = sum;
      }
  }
}
```

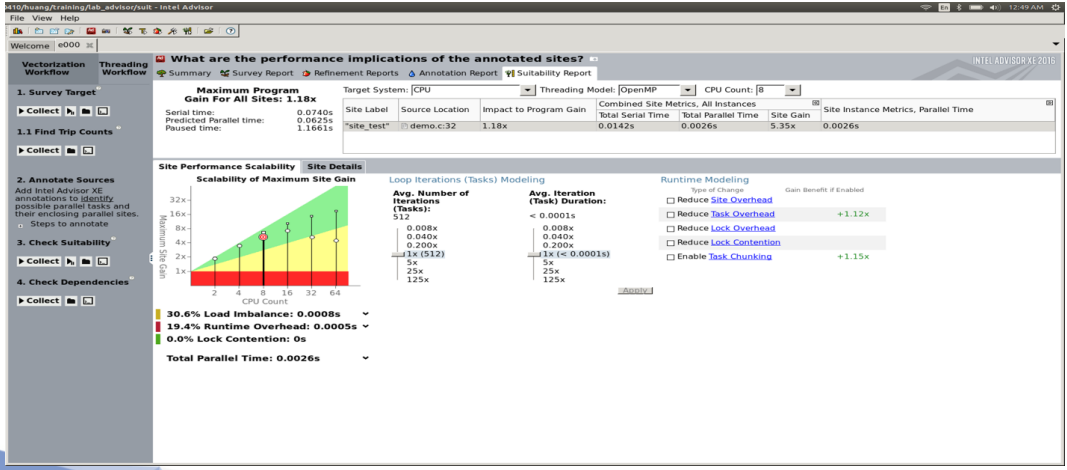
# From serial to annotated to OMP code

```
// matrix multiply routine
void multiply_d(int Size, double **aMatrix,double **bMatrix,
                                                double **product) {
ANNOTATE_SITE_BEGIN(matrix_multiply);
  for(int i=0;i<Size;i++) {
      ANNOTATE_ITERATION_TASK(multiply_task);
      for(int j=0;j<Size;j++) {
        double sum = 0;
        for(int k=0;k<Size;k++) {
          sum += aMatrix[i][k] * bMatrix[k][j];
        }
        product[i][j] = sum;
      }
  }
  ANNOTATE_SITE_END(); }
```

Intel Advisor Reference:

https://software.intel.com/en-us/node/608619

# Advisor Predict mult-thread performance for selected place



# From serial to annotated to OMP code

```
// matrix multiply routine
void multiply_d(int Size, double **aMatrix,double **bMatrix,
                                                  double **product) {
#pragma omp parallel for schedule(dynamic)
  for(int i=0;i<Size;i++) {
      for(int j=0;j<Size;j++) {
        double sum = 0;
        for(int k=0;k<Size;k++) {
          sum += aMatrix[i][k] * bMatrix[k][j];
        }
        product[i][j] = sum;
      }
  }
 }
```

# What Advisor Cannot Do

Intel Advisor is designed to analyze serial code. It does not catch all problems, and it cannot ensure that you have correctly implemented the parallelism! Before deploying your parallel program, you need to test it for dependencies and verify its performance.

## Combine with other available tools

**Intel Inspector** includes a data race and deadlock detection tool that works on the parallel code. It can find more errors because it operates on the parallel code instead of working on the annotated serial code analyzed by the Dependencies tool. Intel Inspector also can find problems with memory: memory leaks, references to freed storage, references to uninitialized memory, and so forth. The memory-checking tool works on serial or parallel code.

**VTune** for hot spot and memory access analysis

# References

Intel VTune Amplifier:
https://software.intel.com/en-us/intel-VTune-amplifier-xe
Intel Advisor:
https://software.intel.com/en-us/get-started-with-advisor
https://software.intel.com/en-us/articles/advisorxe-tutorials
Intel Software Documentation Library
https://software.intel.com/en-us/intel-software-technical-documentation