Profiling your application with Intel VTxune at NERSC
VTune background and availability

• **Focus: On-node performance analysis**
  – Sampling and trace-based profiling
  – Performance counter integration
  – Memory bandwidth analysis
  – On-node parallelism: vectorization and threading

• **Pre-defined analysis experiments**

• **GUI and command-line interface** (good for headless collection and later analysis)

• **NERSC availability (as the vtune module)**
  – Edison (Dual 12-core Ivy Bridge)
  – Babbage (Dual 8-core Sandy Bridge + Dual Xeon Phi)
Running VTune on Edison I

- Use the Cray cc or ftn wrappers for the Intel compilers
- Suggested compiler flags:
  - -g : enable debugging symbols
  - -O2 : use production-realistic optimization levels (not -O0)
- To use VTune on Edison, you have to:
  - Run within a CCM job (batch or interactive)
  - Use dynamic linking if profiling OpenMP code (-dynamic)
  - Use a working directory on a Lustre $SCRATCH filesystem

```bash
edison09:BGW > ftn -dynamic -g -O2 -xAVX -openmp bgw.f90 -o bgw.x
edison09:BGW > mkdir $SCRATCH/vtune-runs
edison09:BGW > cp bgw.x $SCRATCH/vtune-runs/
edison09:BGW > cd $SCRATCH/vtune-runs/
edison09:vtune-runs > qsub -I -q ccm_int -l mppwidth=24 wait ...
```
Running VTune on Edison II

• Once you’re in a CCM job (either interactive or batch script)
  – cd to your submission directory
  – Launch VTune to profile your code on a compute node with aprun

```bash
CCM Start success, 1 of 1 responses
nid02433:~ > cd $PBS_O_WORKDIR
edison09:vtune-runs > module load vtune
nid02433:vtune-runs > aprun -n 1 amplxe-cl -collect experiment_name -r result_dir -- ./bgw.x
```

• `amplxe-cl is the VTune CLI`
  – `-collect`: specifies the collection experiment to run
  – `-r`: specifies an output directory to save results

• **Set OMP_NUM_THREADS and associated aprun options**
  `(-d, -S, -cc depth, -cc numa_node)` as needed

• Results can be analyzed by launching `amplxe-gui` and navigating to the result directory (preferably in NX)
Experiments: General exploration

- Available on Edison and Babbage (SNB + Xeon Phi)

```
nid02433:vtune-runs > aprun -n 1 amplxe-cl -collect general-exploration -r ge_results -- ./bgw.x
```

- Detailed characterization of relevant performance metrics throughout your application
  - Default: low-level detail aggregated into summary metrics
    - Mouse-over for explanation of their significance
    - Can be used to characterize locality issues, poor vectorization, etc.

- Multiple “viewpoints” available:
  - Direct access to hardware event counters
  - Spin / sync overhead for OpenMP threaded regions
Experiments: General exploration

A whole lot of summary metrics!
Experiments: General exploration

- Filter by process and thread ID
- Show loops as well as functions
Experiments: General exploration

Change viewpoint to get to hardware counters, hotspot analysis, and more
Experiments: Memory bandwidth

- Available on Edison and Babbage (Xeon Phi only)
  - Caveat: avoid Babbage SNB for now (node will lock up)

  ```
nid02433:vtune-runs > aprun -n 1 amplxe-cl --collect bandwidth -- -r bw_results -- ./bgw.x
  ```

- Gives DRAM read/write traffic as a function of time during program execution
- Useful to first calibrate with a well-understood code on the same platform (e.g. STREAM)
- Can help determine whether your code is at least partially (effectively) BW bound
Experiments: Bandwidth

Elapsed Time: 73.320s
- CPU Time: 827.786s
- Instructions Retired: 412,366,818,549
- CPI: 5.621

The CPI may be too high. This could be caused by issues such as memory stalls, instruction stalling, branch misprediction, or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

Paused Time: 0s

Average Bandwidth

<table>
<thead>
<tr>
<th>Package</th>
<th>Bandwidth, GB/sec</th>
<th>Read Bandwidth, GB/sec</th>
<th>Write Bandwidth, GB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_0</td>
<td>49.288</td>
<td>29.850</td>
<td>19.430</td>
</tr>
<tr>
<td>package_1</td>
<td>0.088</td>
<td>0.046</td>
<td>0.042</td>
</tr>
</tbody>
</table>

OpenMP Region Duration Histogram

This histogram shows the total number of region instances in your application executed with a specific duration. High number of slow instances may signal a performance bottleneck. Explore the data provided in the Bottom-up, Top-down Tree, and Timeline panes to identify code regions with the slow duration.

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: /stream_c.exe
User Name: sfrench
Operating System: 3.0.101-0.31.1_1.0502.8394-cray_ari_c SUSE Linux Enterprise Server 11 (x86_64) VERSION = 11 PATCHLEVEL = 3

Average BW listed by CPU package
Experiments: Bandwidth

- Click and drag to zoom in for more detail
- OpenMP regions
- Peak BW
- Read, write, and aggregate BW time series
More resources

• At NERSC
  – More details on how to run your analysis on both the Edison compute nodes and the Babbage Xeon Phis
  – Pointers to materials from previous NERSC trainings

• At Intel
  – Main documentation for 2015 version: https://software.intel.com/en-us/node/529213
  – Detailed descriptions of the various experiment types
  – Pointers to tutorials on specific topics or platforms