BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Performance Analysis and Optimization

Samuel Williams

SWWilliams@lbl.gov
Computational Research Division
Lawrence Berkeley National Lab

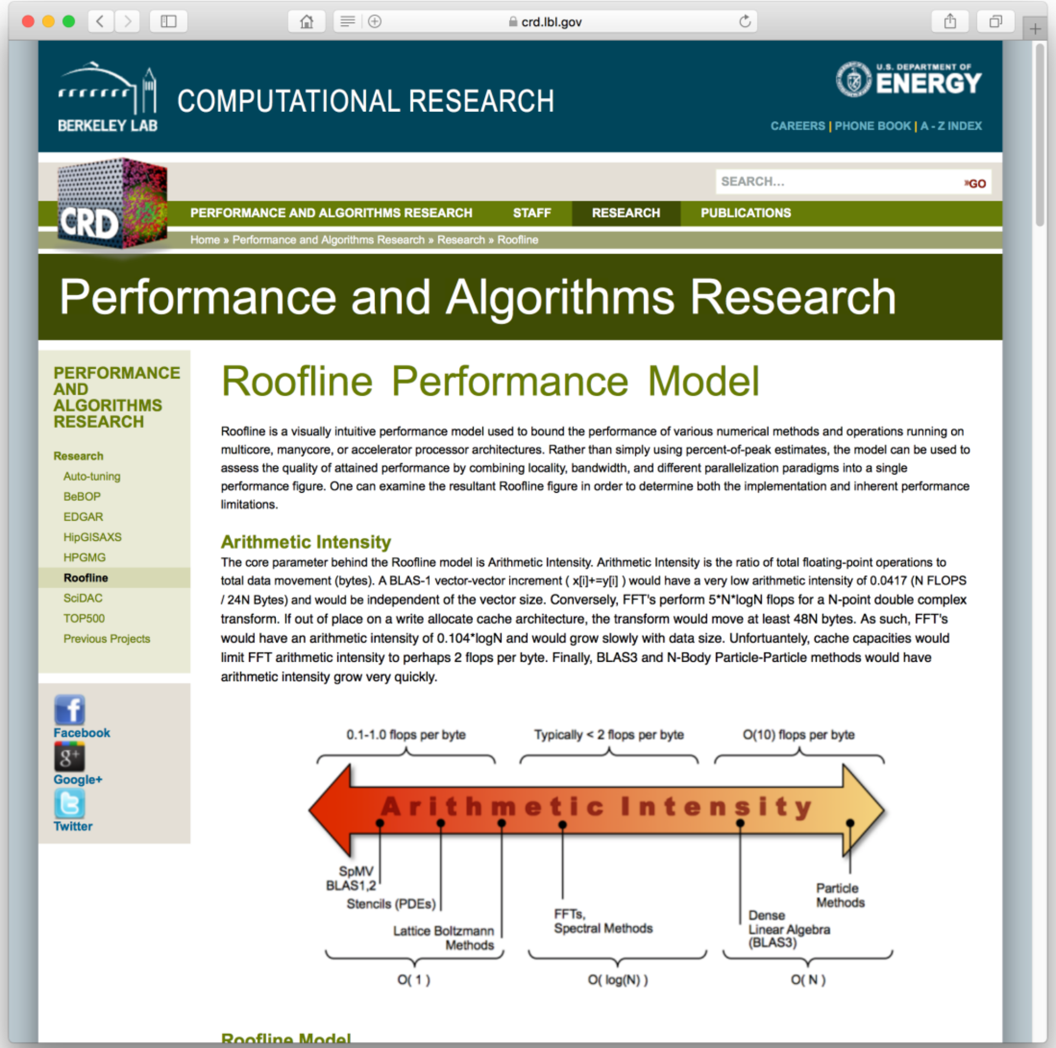
Instrumentation

- Application-specific (manual) instrumentation…
  - Most robust
  - Minimal overhead (omp_get_time)
  - Insensitive to sampling effects
  - Application-specific knowledge can different based on usage (e.g. different levels of MG)
  - High effort / large reward
- Auto-instrumentation (TAU, Advisor, Vtune)…
  - Minimal effort
  - Integrated visualization
  - Sampling effects can confuse performance analysis
  - Using the same function many different ways can confuse analysis
  - Can have high overhead (Advisor/Vtune)

BERKELEY LAB

Roofline Model

- The **Roofline Model** is a throughput-oriented performance model…
  - Tracks rates not time
  - Augmented with Little's Law
    (concurrency = latency*bandwidth)
  - Independent of ISA and architecture
    (applies to CPUs, GPUs, Google TPUs[1], etc…)
- Informs developers which routines are underperforming the processor's capabilities == which routines to optimize



https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

[1]Jouppi et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA, 2017.

BERKELEY LAB

Use by NESAP

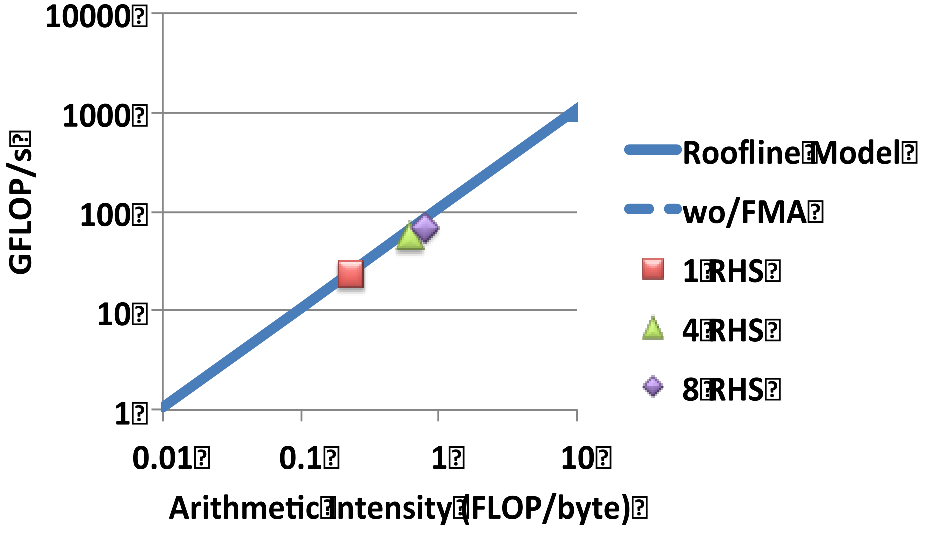
- NESAP is the NERSC KNL application readiness project.
- NESAP used Roofline to drive optimization and analysis on KNL
  - Bound performance expectations (ERT)
  - Use Vtune to quantify DDR and MCDRAM data movement
  - Compare KNL data movement to Haswell (sea of private/coherent L2’s vs. unified L3)
  - Understand importance of vectorization

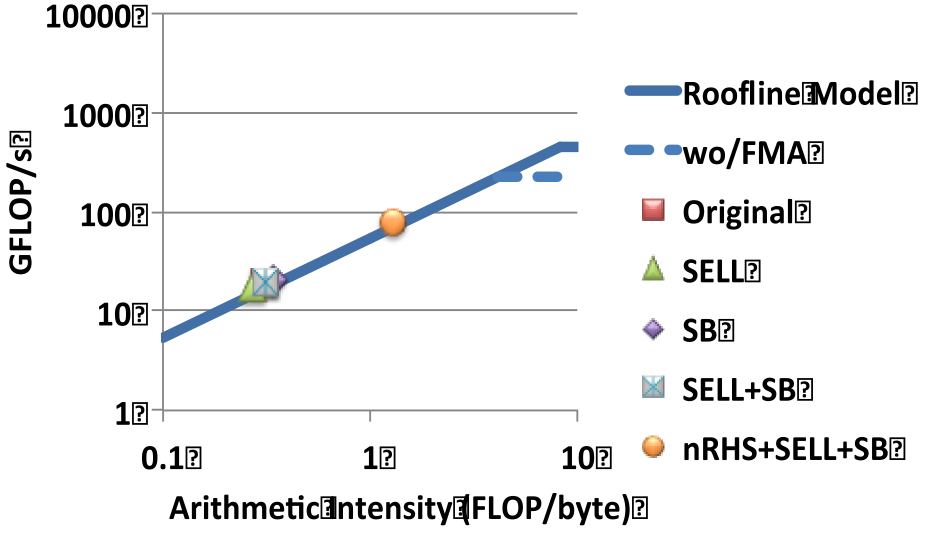
- Doerfer et al., **"Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor",** *Intel Xeon Phi User Group Workshop (IXPUG),* June 2016.
- Barnes et al. **"Evaluating and Optimizing the NERSC Workload on Knights Landing",** *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS),* November 2016.

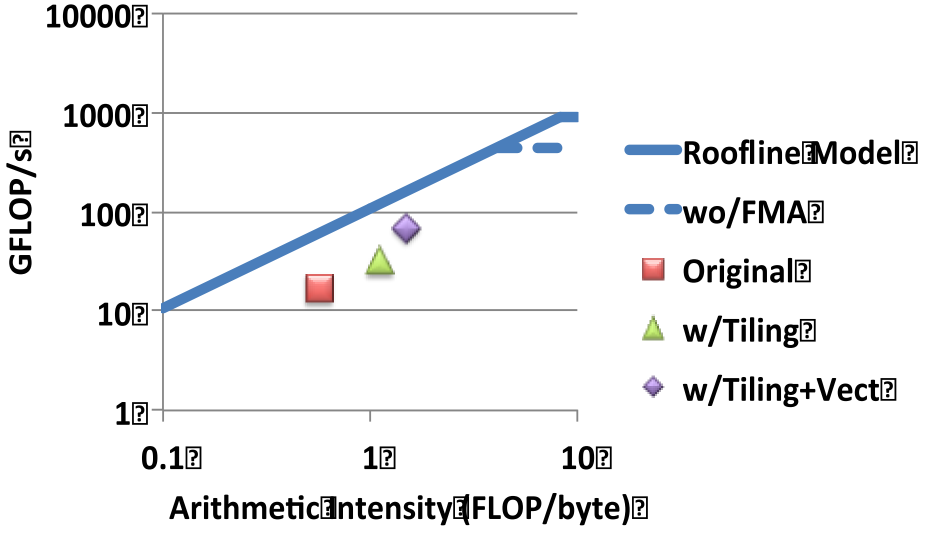
Roofline for NESAP Codes

MFDn | EMGeo | PICSAR

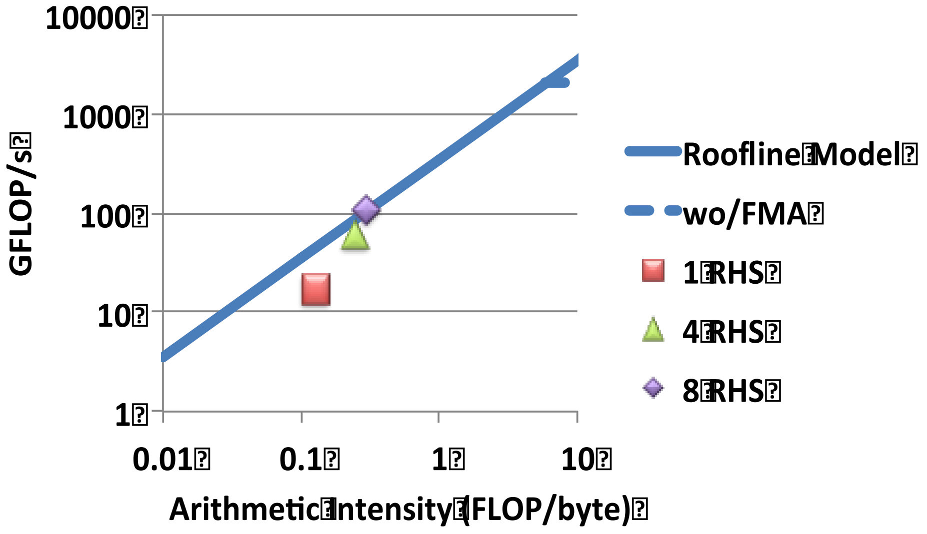
2P HSW

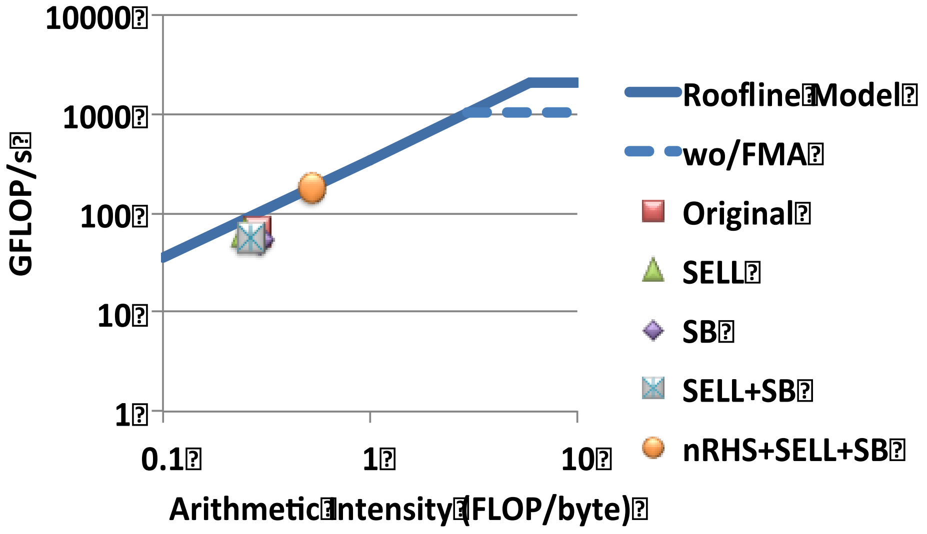


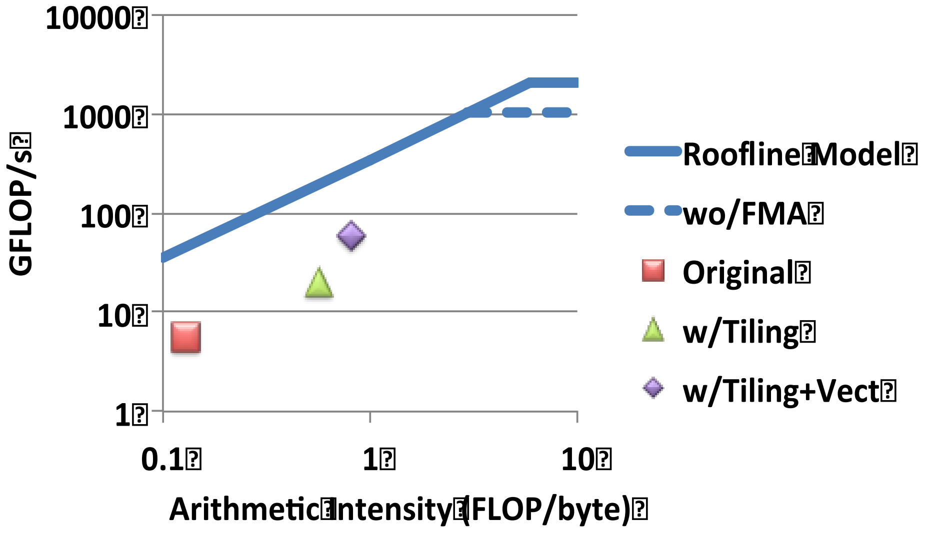




KNL







Intel Advisor

Intel Advisor is a performance analysis tool (evolved from vector advisor)

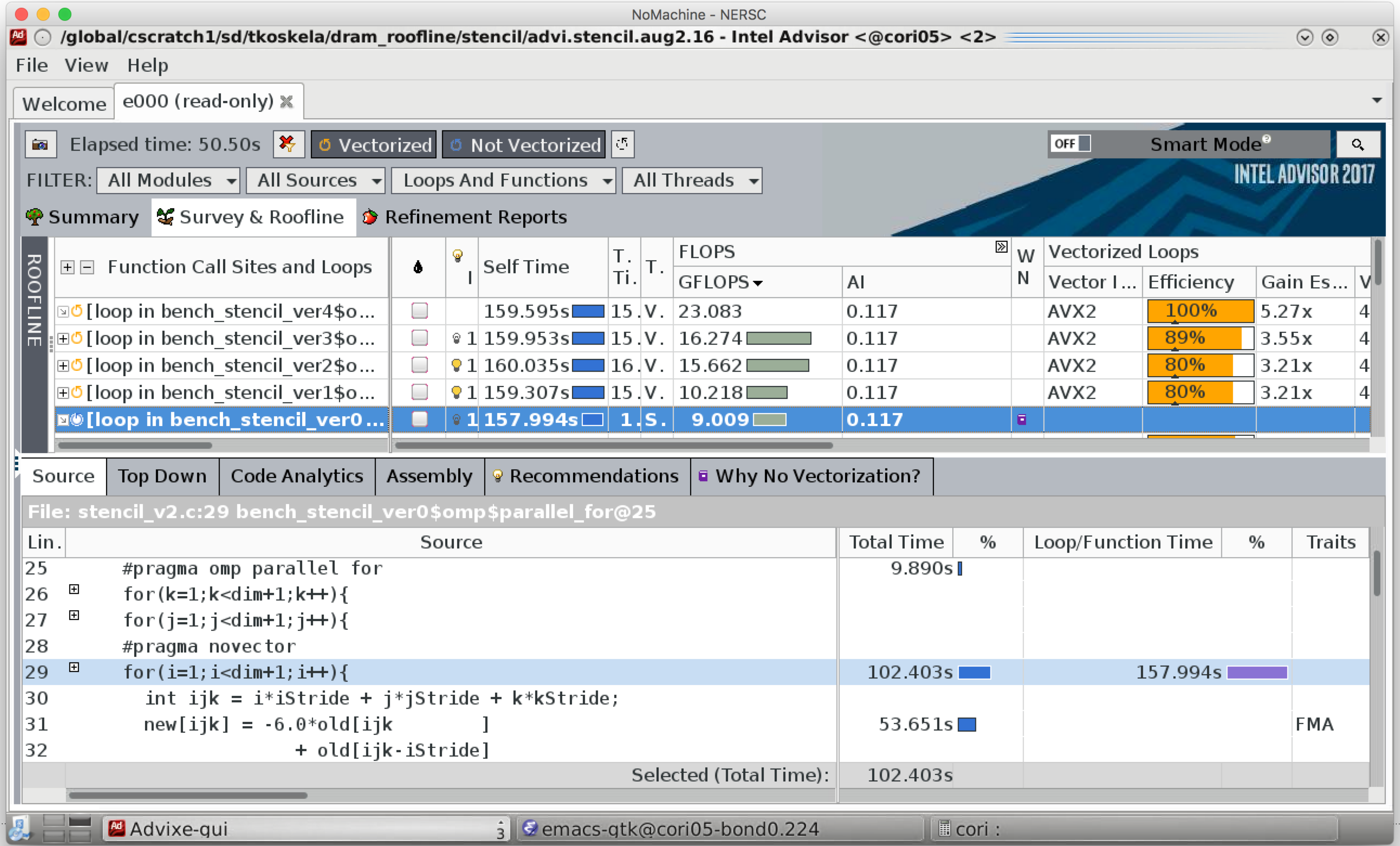
Background

- https://software.intel.com/en-us/intel-advisor-xe
- https://software.intel.com/en-us/articles/getting-started-with-intel-advisor-roofline-feature
- https://www.youtube.com/watch?v=h2QEM1HpFgg

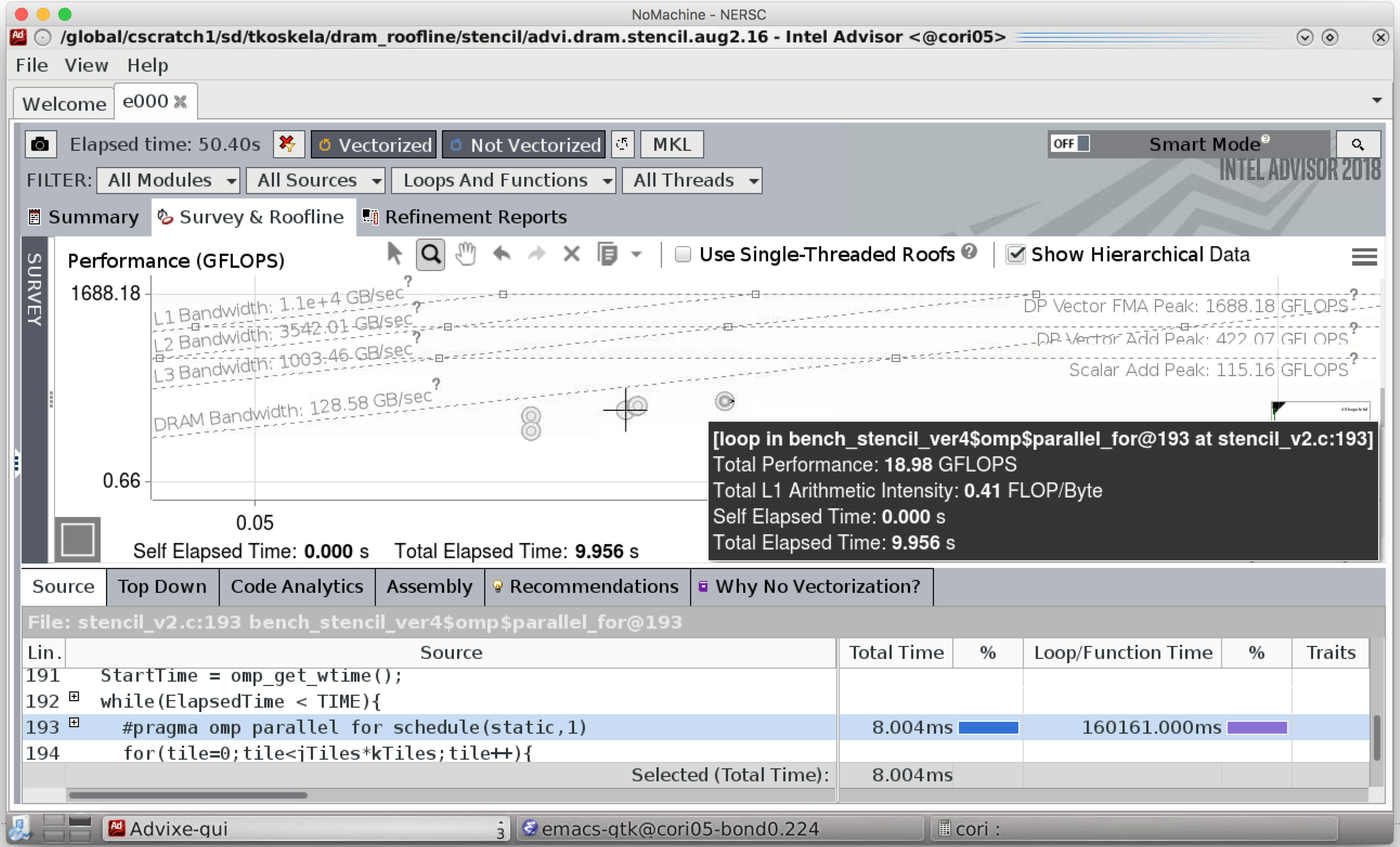
Running Advisor on NERSC Systems

- http://www.nersc.gov/users/software/performance-and-debugging-tools/advisor/

BERKELEY LAB

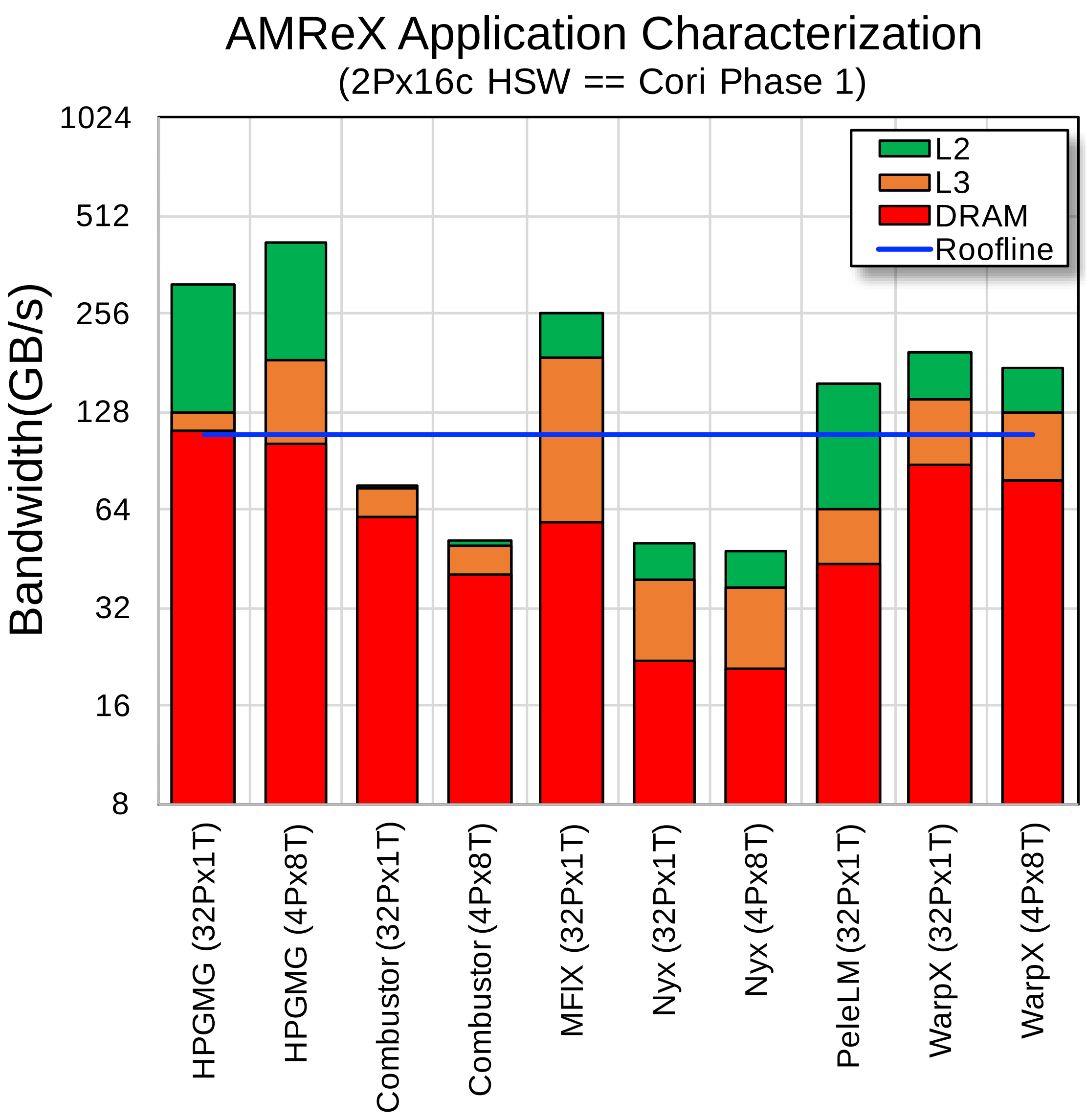


NoMachine - NERSC
/global/cscratch1/sd/tkoskela/dram_roofline/stencil/advi.stencil.aug2.16 - Intel Advisor <@cori05> <2>
File View Help
Welcome
e000 (read-only)
Elapsed time: 50.50s
Vectorized
Not Vectorized
OFF
Smart Mode
INTEL ADVISOR 2017
FILTER: All Modules All Sources Loops And Functions All Threads
Summary
Survey & Roofline
Refinement Reports
ROOFLINE
Function Call Sites and Loops
Self Time
FLOPS
GFLOPS
AI
Vectorized Loops
Vector I...
Efficiency
Gain Es...
[loop in bench_stencil_ver4$o... 159.595s 23.083 0.117 AVX2 100% 5.27x
[loop in bench_stencil_ver3$o... 159.953s 16.274 0.117 AVX2 89% 3.55x
[loop in bench_stencil_ver2$o... 160.035s 15.662 0.117 AVX2 80% 3.21x
[loop in bench_stencil_ver1$o... 159.307s 10.218 0.117 AVX2 80% 3.21x
[loop in bench_stencil_ver0... 157.994s 9.009 0.117
Source
Top Down
Code Analytics
Assembly
Recommendations
Why No Vectorization?
File: stencil_v2.c:29 bench_stencil_ver0ompparallel_for@25
Lin.
Source
Total Time
%
Loop/Function Time
%
Traits
25 #pragma omp parallel for 9.890s
26 for(k=1;k<dim+1;k++){
27 for(j=1;j<dim+1;j++){
28 #pragma novector
29 for(i=1;i<dim+1;i++){ 102.403s 157.994s
30 int ijk = i*iStride + j*jStride + k*kStride;
31 new[ijk] = -6.0*old[ijk] 53.651s FMA
32 + old[ijk-iStride]
Selected (Total Time): 102.403s
Advixe-gui
emacs-gtk@cori05-bond0.224
cori :


NoMachine - NERSC
/global/cscratch1/sd/tkoskela/dram_roofline/stencil/advi.dram.stencil.aug2.16 - Intel Advisor <@cori05>
File View Help
Welcome
e000
Elapsed time: 50.40s
Vectorized
Not Vectorized
MKL
OFF
Smart Mode
INTEL ADVISOR 2018
FILTER: All Modules
All Sources
Loops And Functions
All Threads
Summary
Survey & Roofline
Refinement Reports
SURVEY
Performance (GFLOPS)
Use Single-Threaded Roofs
Show Hierarchical Data
1688.18
0.66
0.05
L1 Bandwidth: 1.1e+4 GB/sec
L2 Bandwidth: 3542.01 GB/sec
L3 Bandwidth: 1003.46 GB/sec
DRAM Bandwidth: 128.58 GB/sec
DP Vector FMA Peak: 1688.18 GFLOPS
DP Vector Add Peak: 422.07 GFLOPS
Scalar Add Peak: 115.16 GFLOPS
[loop in bench_stencil_ver4ompparallel_for@193 at stencil_v2.c:193]
Total Performance: 18.98 GFLOPS
Total L1 Arithmetic Intensity: 0.41 FLOP/Byte
Self Elapsed Time: 0.000 s
Total Elapsed Time: 9.956 s
Self Elapsed Time: 0.000 s Total Elapsed Time: 9.956 s
Source
Top Down
Code Analytics
Assembly
Recommendations
Why No Vectorization?
File: stencil_v2.c:193 bench_stencil_ver4ompparallel_for@193
Lin.
Source
Total Time
%
Loop/Function Time
%
Traits
191 StartTime = omp_get_wtime();
192 while(ElapsedTime < TIME){
193 #pragma omp parallel for schedule(static,1)
8.004ms
160161.000ms
194 for(tile=0;tile<jTiles*kTiles;tile++){
Selected (Total Time):
8.004ms
Advixe-gui
emacs-gtk@cori05-bond0.224
cori :

likwid

- likwid
  - Performance counter infrastructure for x86
  - No sampling (simply reads counters)
  - **<1% overhead** (SDE/VTune/Advisor see >10x)
  - **MPI Scalable** (can run across many processes)
  - Useful for high-level characterization (not connected to soure)
- Now runs on NERSC machines:
  - Verified on Haswell
  - Needs Vtune and likwid modules
  - #SBATCH –perf=vtune
  - Likwid-perfctr
  - Used to characterize AMReX ECP apps…



Threading vs. Processes

- Threads provide no inherent compute advantage over processes
- Threads incur additional overhead (omp parallel, single, barrier, …) == slower in the perfectly parallel world
- Threads provide easy access to shared memory…
  - Some codes are easier to parallelize with threads than MPI
  - Easier to avoid data duplication (memory requirements) with threads than processes
  - Threads can access shared data in cache rather than copying data between processes
  - Using threads simplifies topology-aware MPI process mapping (MPICH_RANK_REORDER is often insufficient)… topology-aware is still important on Aries/Dragonfly and IB/FatTrees
  - Using threads provides on-ramp to GPUs and other emerging architectures

Threading Experiments/Analysis

- Fix process concurrency, increase threading …
  - Ideally, function runtime should scales as O(1/omp_num_threads)
  - Identify functions that plateau (saturation)
  - Identify functions that are flat (sequential bottlenecks)
  - Identify functions that increase (duplicated work… e.g. use of Fortran's sum(a(:)))
- Fix hardware concurrency (cores), increase threads while reducing processes…. (32x1, 16x2, 8x4, 4x8, 2x16, 1x32)…
  - Distinguish true flat MPI (no –fopenmp) from hybrid w/1 thread (-fopenmp + OMP_NUM_THREADS=1) == threading overhead
  - Identify functions that are flat, better with threads, or better with processes
  - Especially useful for communication routines

BERKELEY LAB

KNL-specific issues

- GNU and Intel runtimes treat OMP_PLACES/PROC_BIND differently…
  - use same compiler for all threaded routines / know which settings to use
  - (I use KMP_AFFINITY with Intel)
- If you can fit in 16GB, run in quadflat…
  - avoids cache aliasing issues where some nodes are slower than others
  - Can manifest as abnormally high MPI_Wait times and degraded scalability
  - Use srun … numactl –m1 ./a.out … (or use –p1 or no numactl but allocate key data in HBM)
- use large contiguous blocks with 2M pages for data and MPI buffers…
  - Better performance for complex (less streaming) memory access patterns
  - minimizes NIC TLB pressure == higher MPI bandwidth
- Examine vectorization reports…
  - KNL is very slow if not vectorized

Questions