Empirical Roofline Toolkit (ERT)

Brian Van Straalen

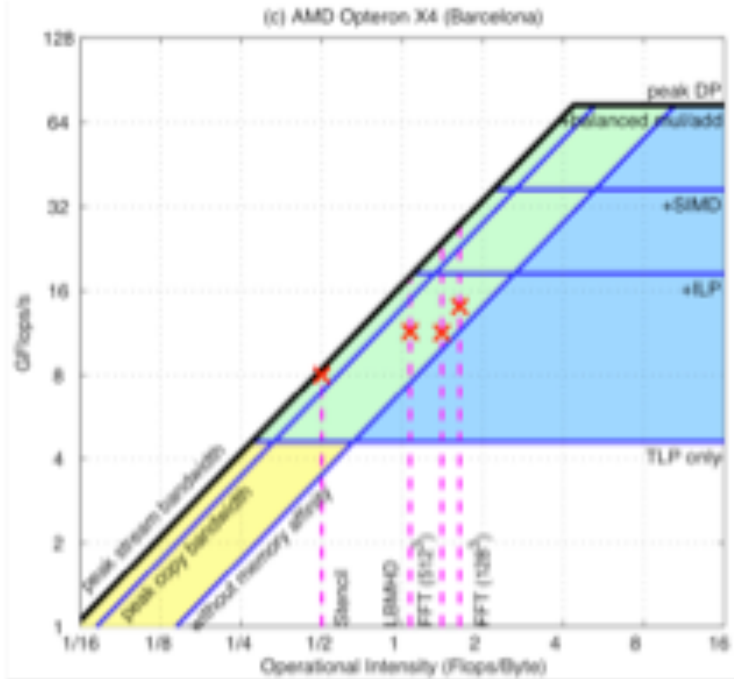
Terry Ligocki, Linda Lo, Wyatt Spear, Matt Cordery,
Sam Williams, Leonid Oliker, Nick Wright

Lawrence Berkeley National Laboratory

BVStraalen@lbl.gov

Review: The Roofline Model

- The Roofline model provides an intuitive model and figure for understanding kernel performance on various architectures.
- Unfortunately, the approach suffers from three factors….



- "***Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures***", **Williams, Waterson, Patterson. 2008**

Rooflines are hard to build

1. **HW Characterization:** Construction of the roofline model requires expert knowledge of the target processor microarchitecture (ILP, TLP, DLP, issue policies, cache/memory capacities/bandwidths, …). This can be hard to come by (limited documentation and requires experts to convert into the model).
2. **Execution Monitoring:** In order to read the figure, one must know the associated characteristics of the kernel. Today, performance characterization of kernels often degenerates into just run time. We need to know #flop's, SIMDization rates, ILP, TLP, actual data movement. On most machines, performance counters often fail to accurately report DRAM data movement (BGQ's HPM is the only success thus far).
3. **SW Characterization:** We need a target of what is theoretically possible for a kernel. This requires an expert in the architecture and the algorithm to predict the performance of the HW/SW stack when compiling/running this routine.

Instead of an Oracle, why not try to assess the realizable Roofline *empirically* with a suite of suitably transparent benchmarks?

- Empirical Roofline Toolkit (or ERT).

Roofline Toolkit

- DOE SciDAC Institutes FastMath (Math Algorithms) and SUPER (Performance Computing) have been developing new algorithms which are more compute intensive in order to avoid the memory bottleneck.

- However, they need to know the characteristics of architectures (today and in the future) in order to bound how aggressive their new algorithms should be.

- To that end, SUPER and FastMath wrote a white paper to fund a Roofline augmentation for SUPER
- This involved the creation of a **Roofline Toolkit** that automates the construction of theoretical and execution Roofline models and figures.

- The funded Roofline augmentation was split among LBL, U Oregon, and Argonne National Lab.

Roofline Toolkit

- The Roofline Toolkit has four components

- This talk will primarily focus on the ERT
  - designed to be a portable machine characterization framework
  - implemented as MPI+OpenMP in order to verify users can run hybrid implementations correctly (i.e. correct mpirun/aprun/… affinity options)
  - variable working set designed to highlight cache hierarchy
  - variable compute intensity kernels
  - produces a JSON file for database and visualization
  - internal release in November, 2014
  - external release in March, 2015

Empirical Roofline Toolkit (ERT) (LBL/CRD)	Performance Counter Triage (LBL/NERSC)	Application Characterization (ANL)
Visualization (University of Oregon)		

Simple benchmarks

Bandwidth

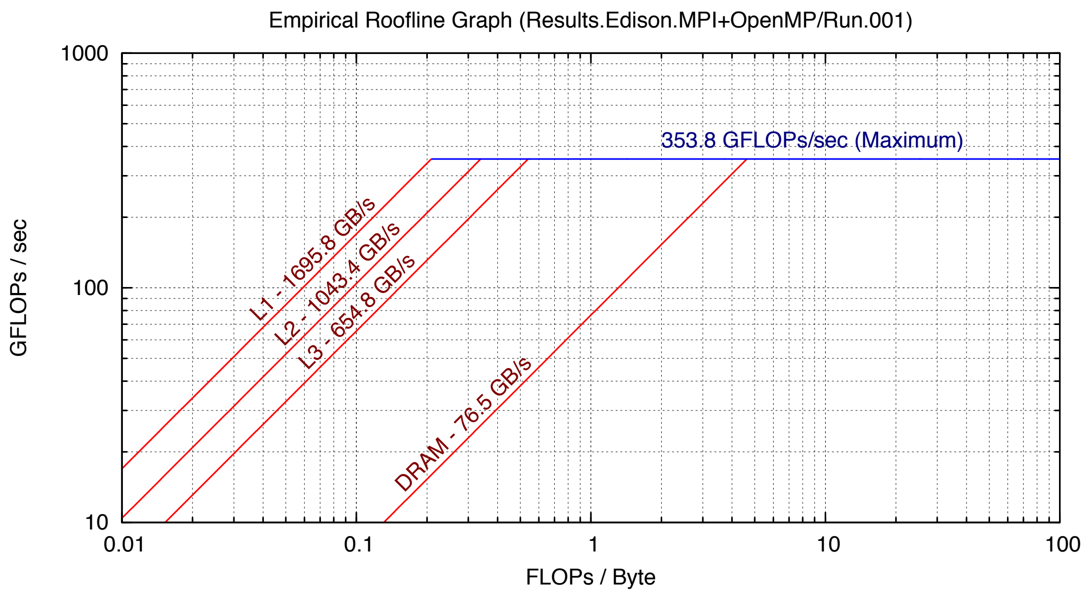
```
void Kernel (uint64_t size, unit64_t
trials, double * __restrict__ A) {
  double alpha = 0.5;
  uint64_t i, j;
  for (j = 0; j < trials; ++j ) {
    for (i = 0; i < nsize; ++i) {
      A[i] = A[i] + alpha;
    }
    alpha = alpha * 0.5;
}}
```

GFlops

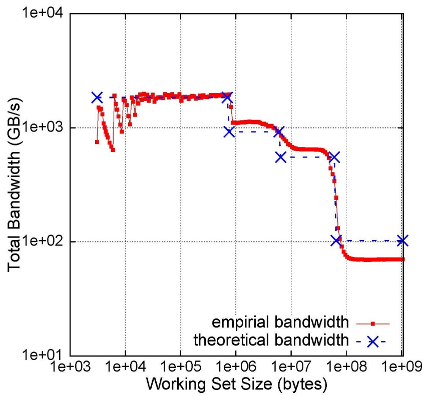
```
void Kernel (uint64_t size, unit64_t trials,
double * __restrict__ A) {
  double alpha = 0.5;
  uint64_t i, j;
  for (j = 0; j < trials; ++j ) {
    for (i = 0; i < nsize; ++i) {
      double bete = 0.8;
     #if FLOPPERITER == 2
      beta = beta * A[i] + alpha;
     #elif FLOPPERITER == 4
      beta = beta * A[i] + alpha;
      beta = beta * A[i] + alpha;
     #elif FLOPPERITER == 8
      …
     #endif
      A[i] = beta;
    }
    alpha = alpha * 0.5;
}}
```

ERT Results

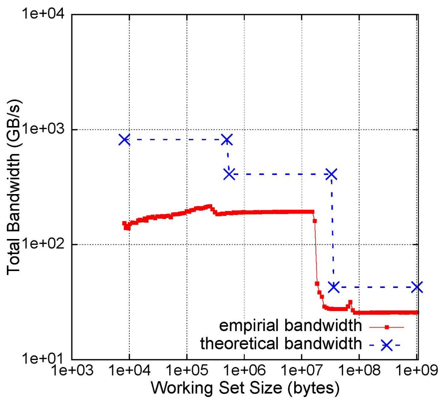
- Consider Edison (2P IVB) with MPI+OpenMP…
  - theoretical flops = 460 GFlop/s
  - theoretical L1 = 1.8TB/s (1:1)
  - theoretical DRAM = 102 GB/s
- The attained performance departs slightly from the theoretical performance but is a better measure of what the computing system (Processor + Compiler + Runtime) can deliver on real applications.
- Similar experiments have been run on BGQ and MIC and are now being automated in the ERT.
  - Results on BGQ suggest…
    - write-thru L1 cache
    - good compilers
    - need for TLP
  - Results on MIC required extreme AI to attain peak



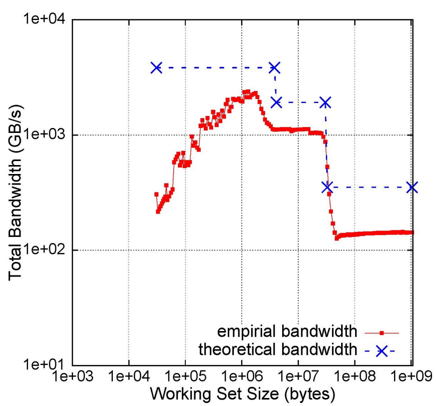
Edison (Intel Ivy Bridge CPU)



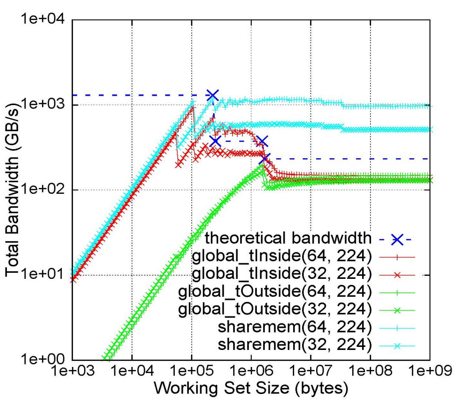
Mira (IBM Blue Gene/Q)



Babbage (Intel Xeon Phi)

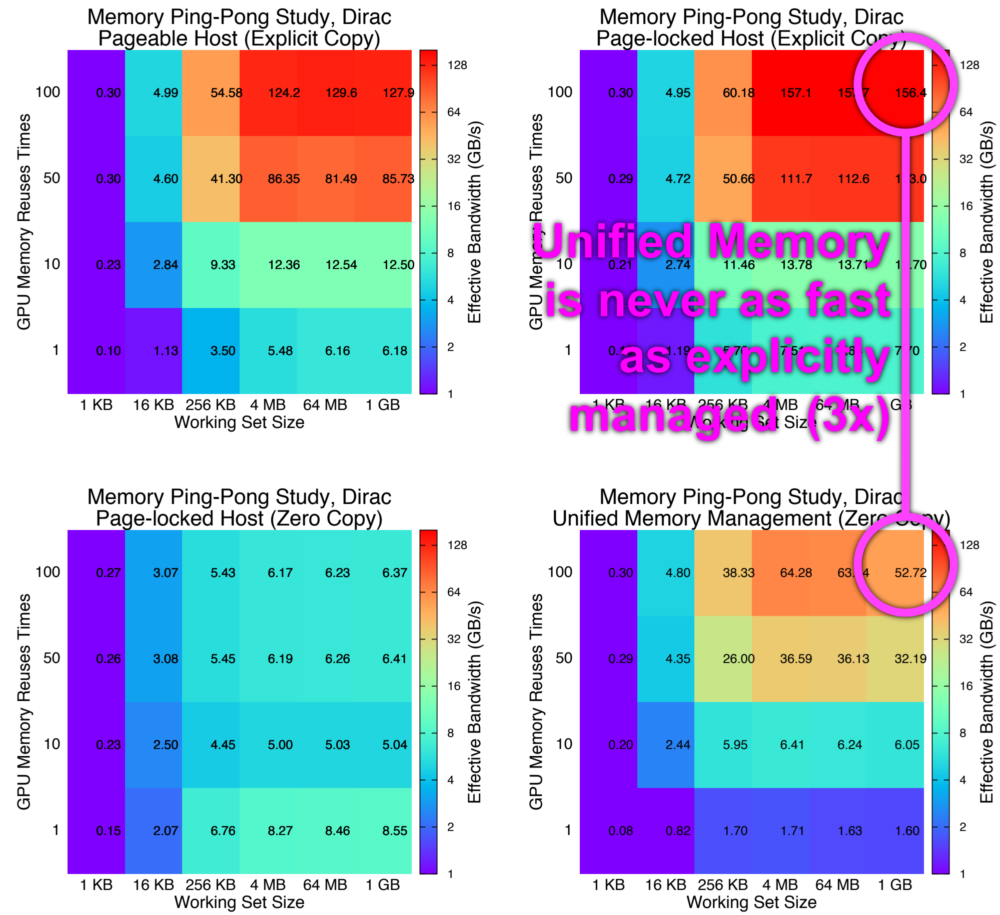


Titan (Nvidia K20x)



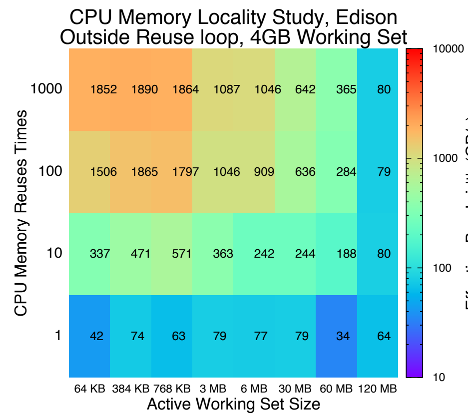
CUDA Unified Memory

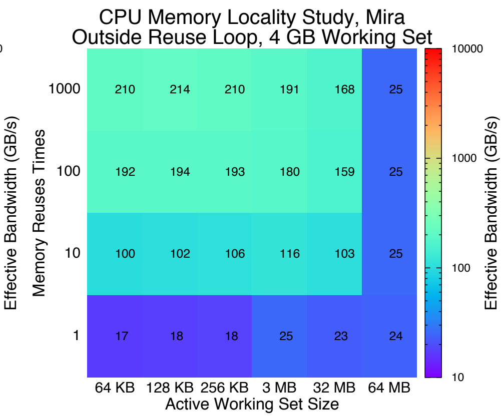
- CUDA is continually evolving
- Older versions of CUDA required user managed copying of data to/from the device…
  - pageable (malloc)
  - pinned memory (cuda malloc)
- Recently, CUDA introduced…
  - Unified Virtual Addressing
  - Zero Copy Memory
  - Unified (managed) Memory
- These push the mismanagement of data locality into the driver.
- Programmers need automated technologies to characterize performance as a function of …
  - memory allocation
  - spatial locality
  - temporal locality
  - interconnect (PCIe vs NVLINK)
- Yu Jung (Linda) Lo, et al, PMBS’14

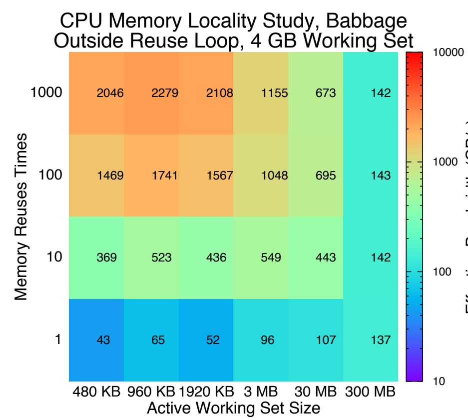


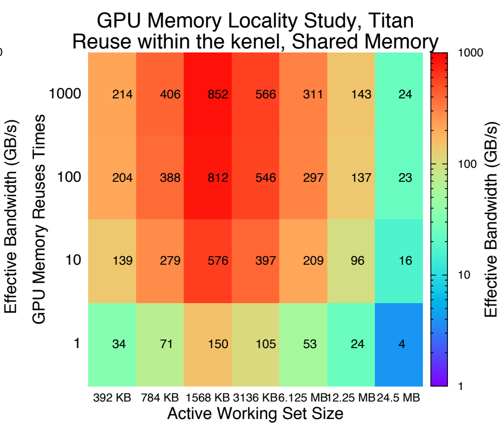
Temporal vs. Spatial Locality vs. Threading Costs

- We can extend the same spatial vs. temporal experiments conducted for CUDA unified memory to any cache hierarchy.
- We may define…
  - large total working set (**TWS**) to flush caches
  - active working set (**AWS**) = spatial locality
  - reuse (**temporal locality**)
- Additionally, we can capture thread synchronization costs…
  - coarse-grained synchronization (synchronize once per AWS)
  - fine-grained synchronization (once per AWS update)
- We can also run a similar code on the GPUs to measure their cache behavior…
  - interestingly 'global' memory dramatically underperforms conventional wisdom for small problems
  - 'shared' memory attains reuse in the L2 for large problems (better than GDDR BW)
  - must exploit 'shared' memory to get good performance for small problems (terrible for large)
- **Original Roofline BW ceilings were basically the top row**









Visualization and Eclipse Integration

- Wyatt Spear / UO

- Roofline charts implemented in JavaFX.

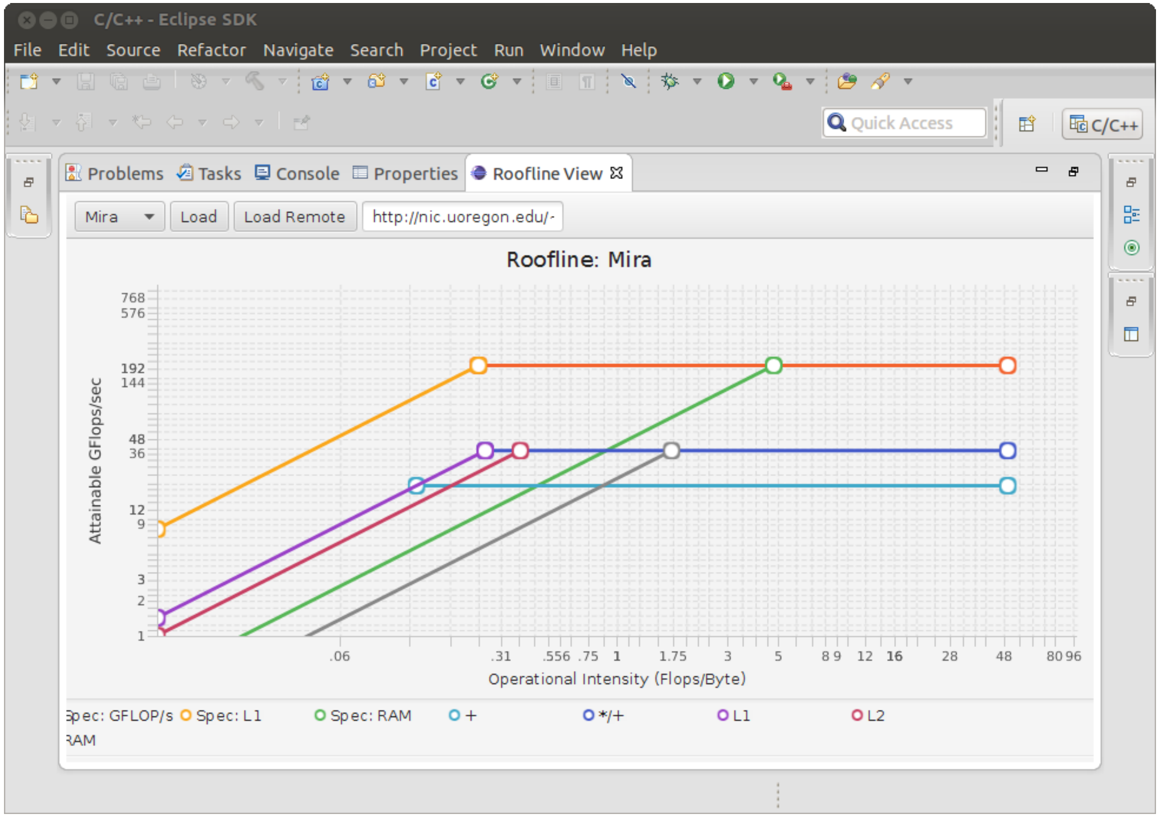
  Allows for portable, standalone viewer

- Roofline data is stored in JSON files
  - performance metrics and metadata
  - facilitates search/comparison between trials, systems and benchmarks

- Remote database for community access to Roofline data

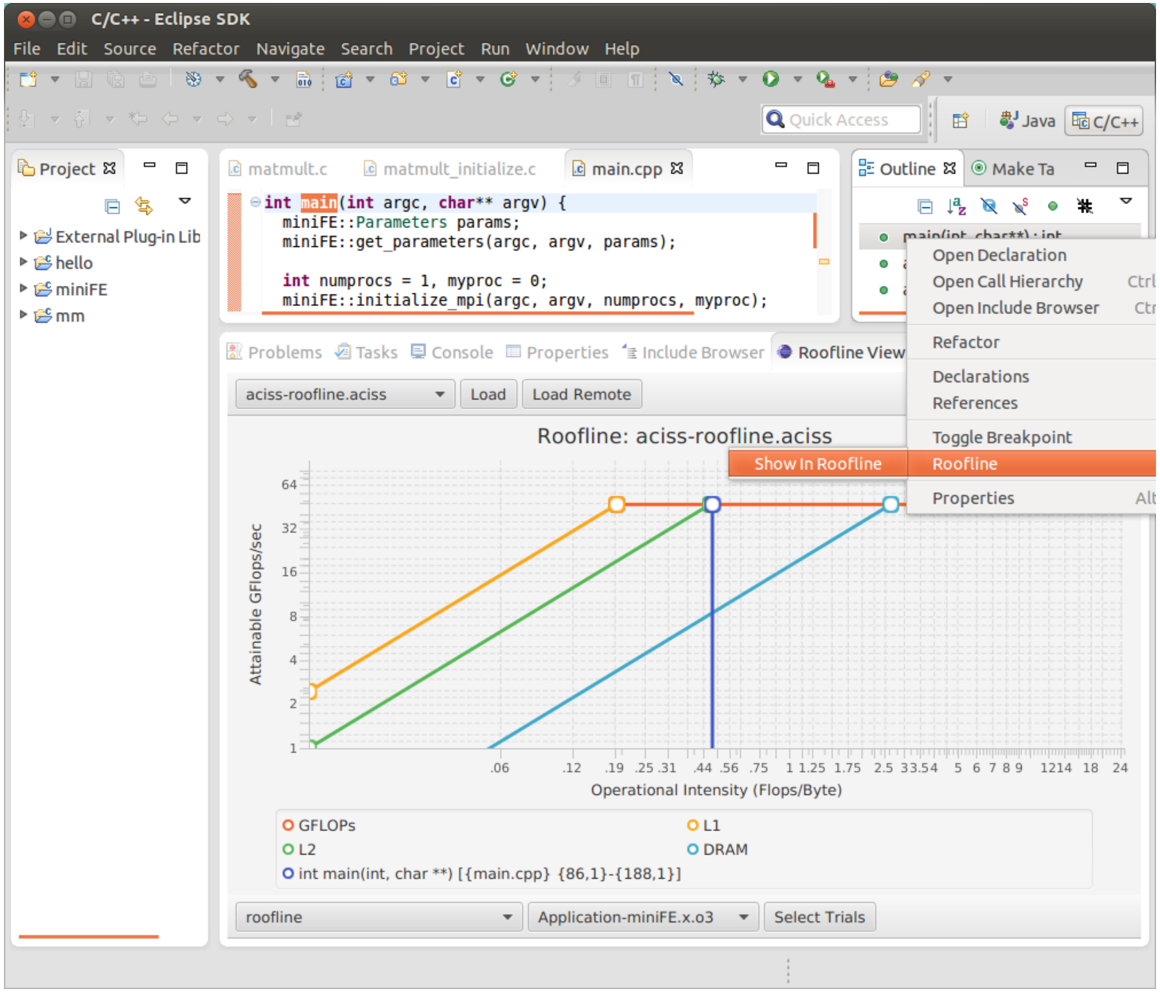
Roofline UI

- Roofline UI can real data from remote repo or local disk
- quick/easy selection from multiple data sets
- values are shown on mouse-over (more precise on log-log)



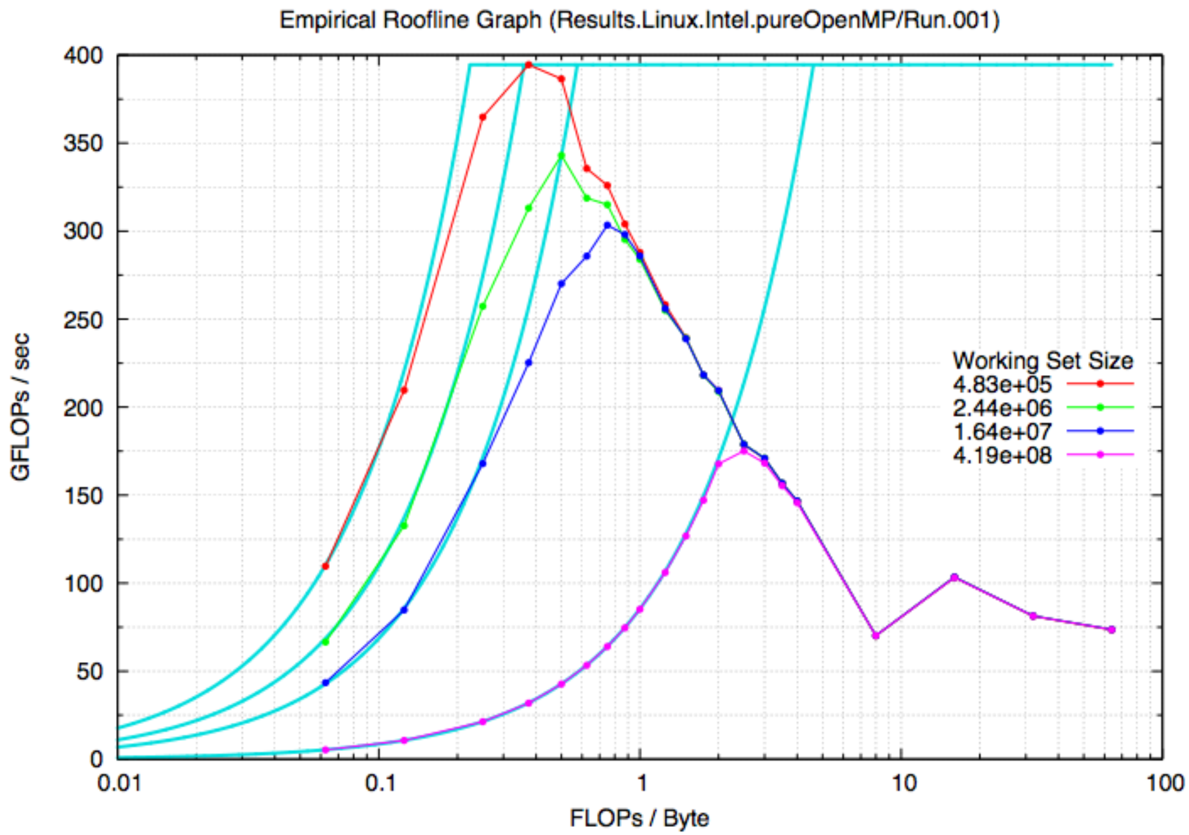
Under Development… Roofline-Eclipse Integration

- Select application events in Eclipse source outline
- Display values from TAUdb database on Roofline chart
- *Not yet part of official release*



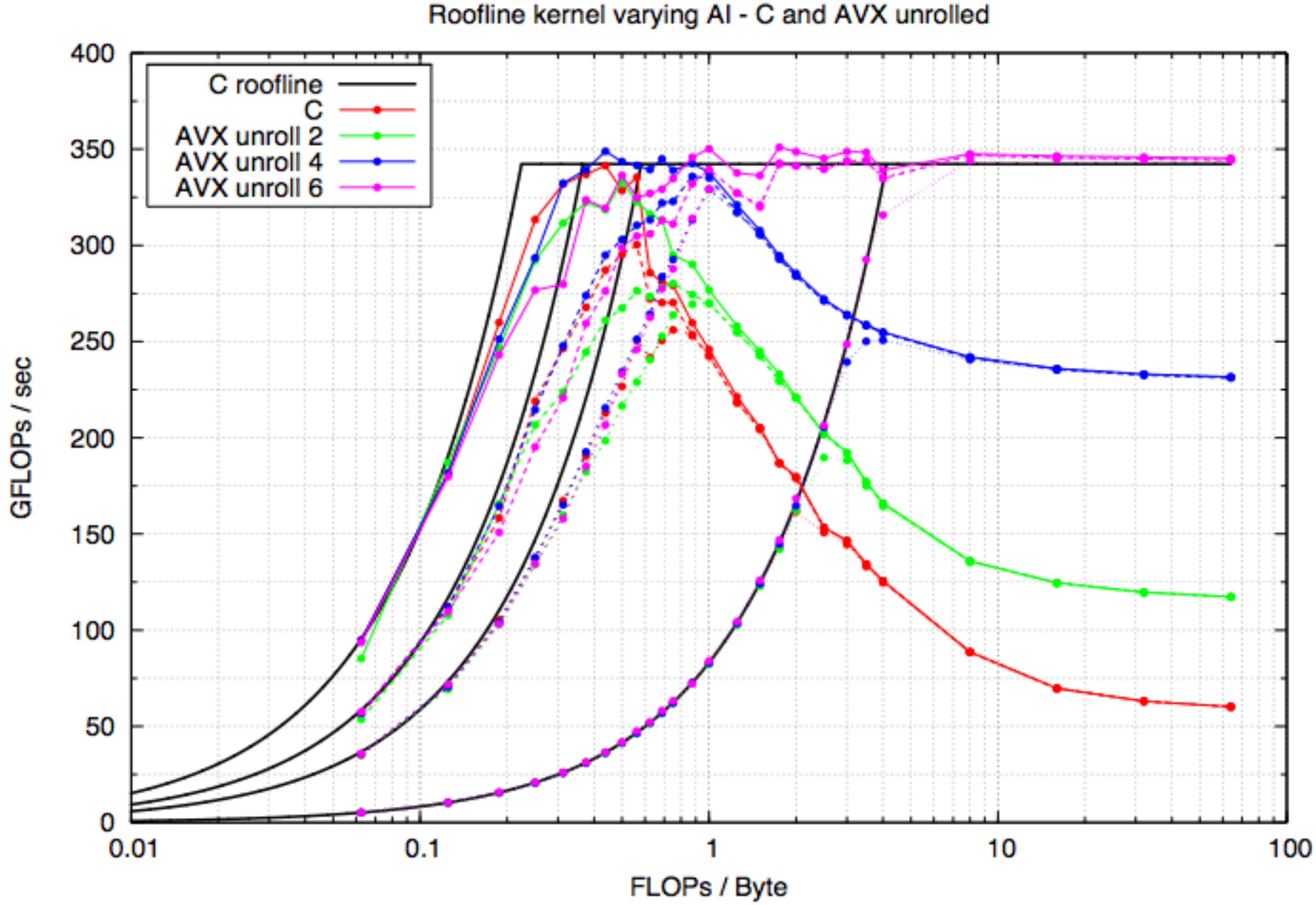
Storm on the Horizon

❖ The data sets hiding behind the Roofline key metrics offer a view of a darker future



- icc compiler stops vectorizing. Fine, I'll vectorize it myself……

Another bonk….

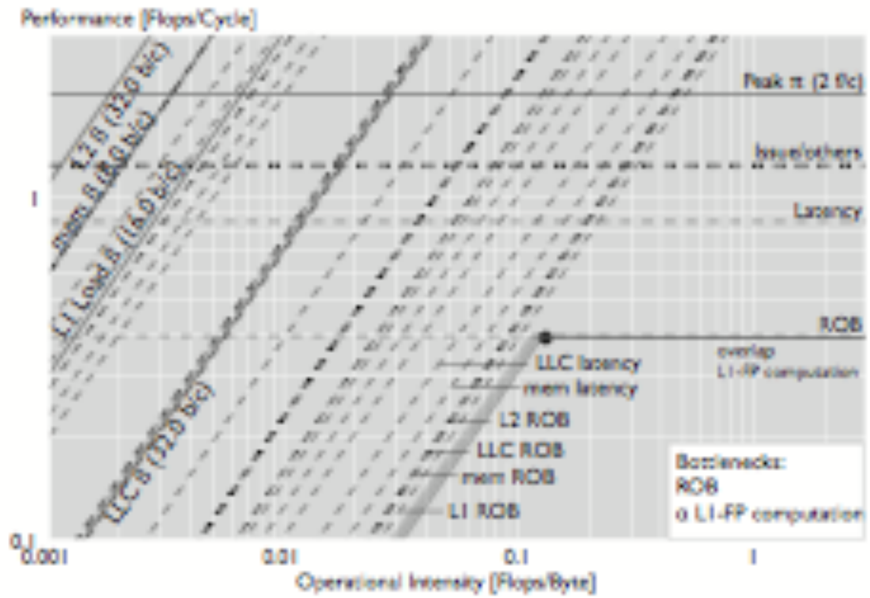


Manually vectorizing means you are now unrolling by hand

- The Out-of-Order limitations now kick in. (ROB limits).

What to do about it?

- "*Extending the Roofline Model: Bottleneck Analysis with Microarchitectural Constraints*" Cabezas & Püschel (ETH). 2014
  - Roofline specific to algorithm
  - Microarchitecture emulated



(b) Roofline plot for FFT, size 1048576, warm cache.

- 2 parameter Roofline cartoon needs to become higher dimension
  - Needs register memory line
  - Things to be quantified: Out-of-Order, load/store slots, launch, synch.
  - 2D slices of Roofline will likely remain illustrative

Beyond The Roofline Model

- Roofline is a streaming performance model…
  (presumes bandwidth dominates latency and overhead)
- Roofline assumes a simple locality model (giant, data-parallel working sets)
- Although these assumptions are often true today, they may not hold in the future…
  - although applications teams will still weak-scale with respect to the number of nodes, many want to strong-scale with respect to he number of cores on a node in the future. (i.e. keep the problem size per node fixed, but replace frequency scaling with multithreading).
  - The cost of thread/device synchronization increases (<1us on Edison, >10us on MIC/GPUs) with parallelism.
  - Real applications have complex locality patterns with finite working set sizes and limited reuse at each level.

- **We need to examine ways of extending the Roofline model to capture the effects of high synchronization costs and complex locality patterns and present the performance implications concisely**

Summary and Next Steps

- We have an initial public release of the ERT available for download

- We are actively collaborating on application characterization (theoretical and empirical) and visualization.
  - visualizer can plot roofline data from database
  - continued Eclipse integration
  - suggestions on analysis and performance counters are welcome

- We plan on generalizing the locality vs. synchronization benchmarks and including them in the ERT.
  - what else are we missing?