Technology Translation: Modeling, Measurement and Analysis

SUPER — INSTITUTE FOR SUSTAINED PERFORMANCE, ENERGY, AND RESILIENCE

Philip C. Roth, Jeremy S. Meredith, Jeffrey S. Vetter
Oak Ridge National Laboratory

Samuel Williams, Brian Van Straalen, Terry Ligocki, Leonid Oliker, Matt Cordery, Nick Wright
Lawrence Berkeley National Lab

Yu Jung (Linda) Lo
University of Utah

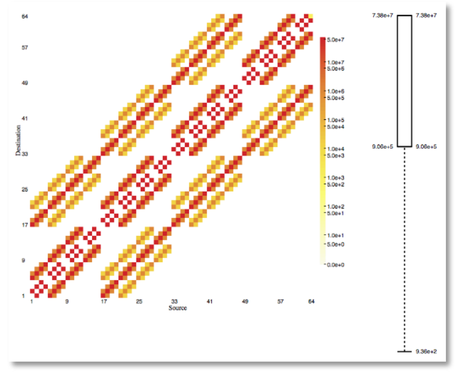
Wyatt Spear, Boyana Norris, Allen Malony, Sameer Shende, Kevin Huck, Nick Chaimov
University of Oregon

Heike McCraw, Asim Yarkhan, Sangamesh Ragate
University of Tennessee

Shirley Moore
University of Texas at El Paso

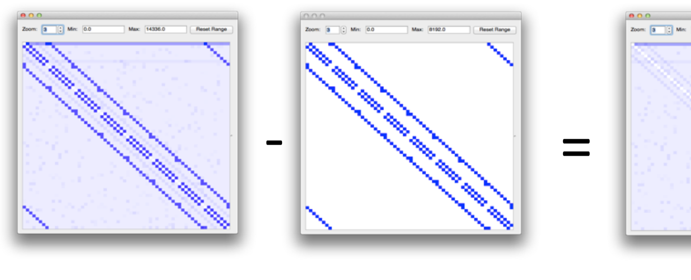
U.S. DEPARTMENT OF ENERGY | Office of Science

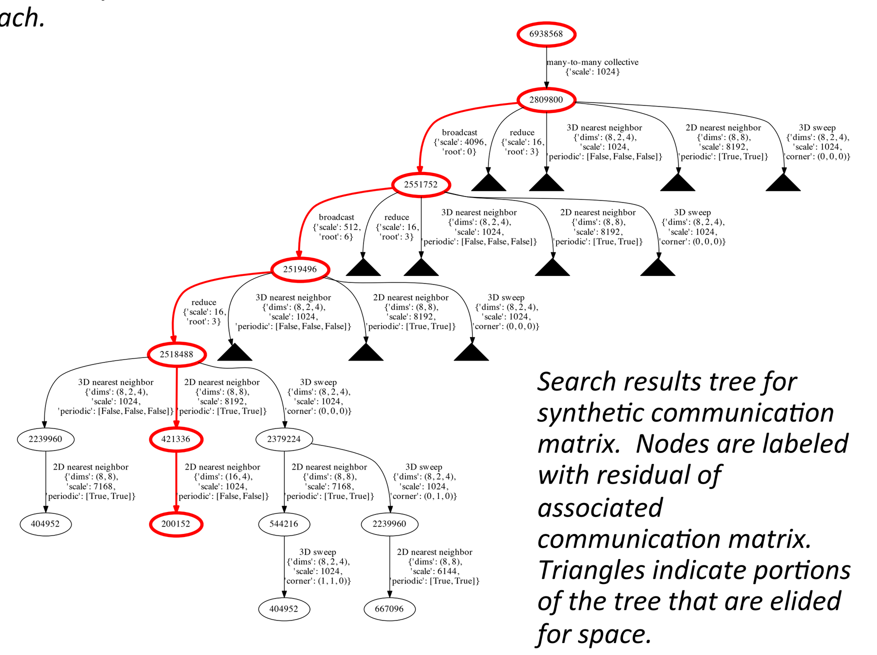
Automated Characterization of Message Passing Application Communication Patterns



- **The Problem**
  - We want a concise way to express application communication demands
  - E.g., "3D Nearest Neighbor, broadcast, and reduce" instead of communication matrices
  - But…strong expertise needed to identify patterns from communication matrices
- **Our Approach**
  - Automated search using a library of patterns to identify collection of parameterized patterns that best explains observed communication
  - Adopts idea from astronomy's sky subtraction: given an image, remove the known to make it easier to identify the unknown
  - Input is communication matrix, e.g., as collected by the Oxbow version of mpiP (http://oxbow.ornl.gov)
  - Each search step involves recognizing a pattern, scaling the recognized pattern as large as possible, and removing the scaled pattern to produce a communication matrix containing as-yet-unrecognized communication behavior

*Recognizing and removing the contribution of a 2D nearest neighbor pattern in a synthetic communication matrix. This represents **one step** in a search-based approach.*

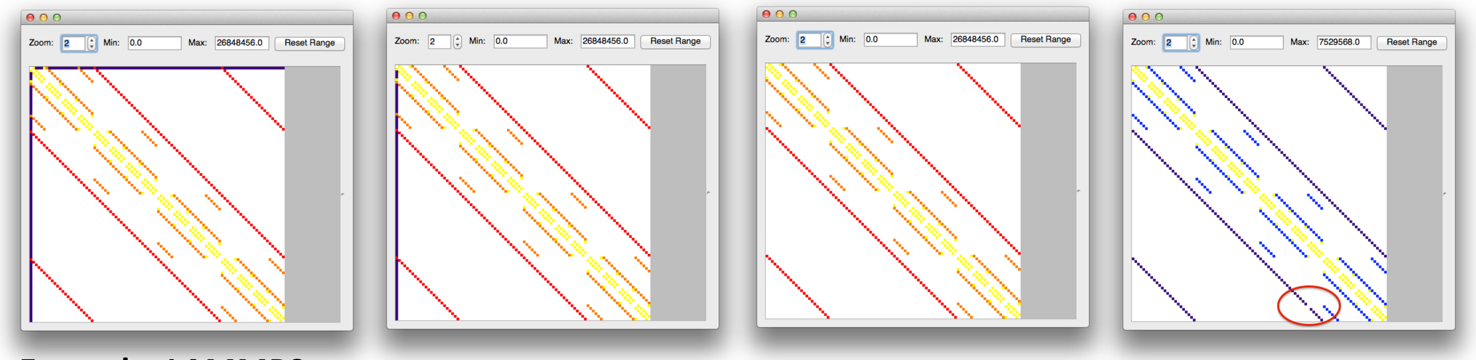




Search results tree for synthetic communication matrix. Nodes are labeled with residual of associated communication matrix. Triangles indicate portions of the tree that are elided for space.

- **Search Results Tree**
  - Communication matrices at nodes
    - Initial communication matrix associated with tree root
  - Recognized, parameterized patterns label edges
  - Child node's matrix is result of subtracting recognized pattern from parent's matrix
  - When child node is added to tree, recursively apply search starting at the child
- **Output**
  - *Residual* of matrix is total amount of communication volume represented in matrix
  - When search completes, path from root to leaf with smallest residual indicates collection of patterns that best explain original matrix (red path in example search results tree)
  - Output from the automated search is a list of parameterized patterns that best explain input communication matrix
  - Output is trivially converted into expression with parameterized patterns as terms, e.g.:

$$C_{LAMMPS} = 13354 \cdot Broadcast(root:0) + 700 \cdot Reduce(root:0) + 19318888 \cdot 3DNearestNeighbor(dims:(4,4,6), periodic:True)$$

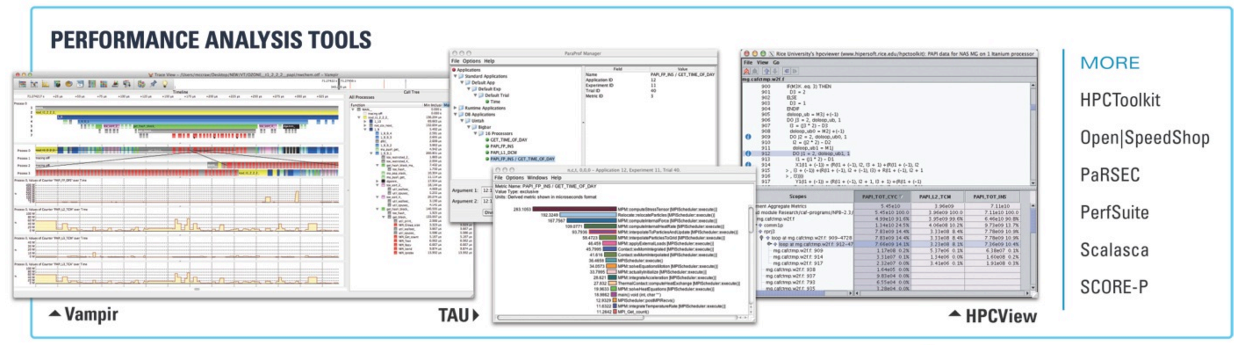


Original matrix — *After removing broadcast* — *After removing reduce* — *After removing 3D nearest neighbor, dimensions (4,4,6), periodic*

- **Example: LAMMPS**
  - Communication matrix collected using Oxbow's mpiP for 96-process LAMMPS run of EAM benchmark on Keeneland Initial Delivery System
  - Basically a 3D Nearest Neighbor pattern, but detected as imperfect (red circle in last figure)
- **Pilot implementation:** Python-based using NumPy and SciPy matrix support
  - Pattern recognizers/generators are Python classes
    - Many-to-many, Broadcast, Reduce, 2D Nearest Neighbor, 3D Nearest Neighbor, 3D Wavefront (sweep) from a corner, Random (generate only)

Performance API – PAPI

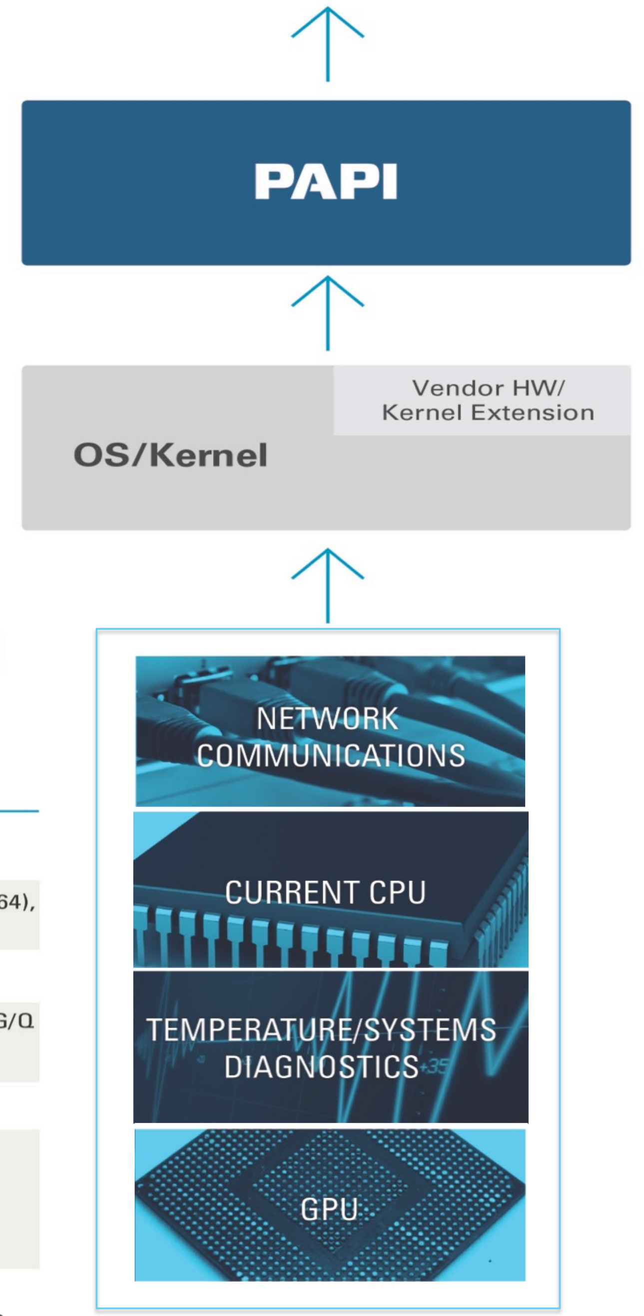
PAPI (Performance Application Programming Interface) provides the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. In addition, it provides access to a collection of components that expose performance measurement opportunities across the hardware and software.



PAPI FEATURES

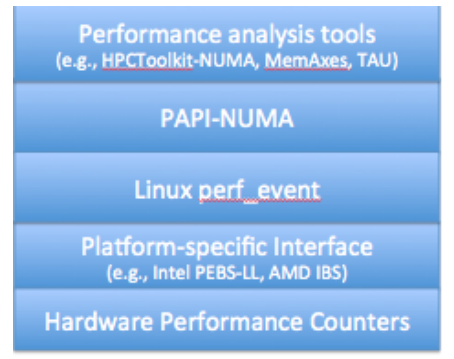
- Standardized Performance Metrics
- Easy Access to Platform-Specific Metrics
- Multiplexed Event Measurement
- Dispatch on Overflow
- Overflow & Profiling on Multiple Simultaneous Events
- Bindings for C, Fortran and Matlab
- User Definable Metrics derived from Platform-Specific Metrics
- Support for Virtual Computing Environments

NEW FEATURES
- Performance counter monitoring at task granularity for dataflow runtime PaRSEC
- Measure Component level power
- Set power bounds within a HPC execution via PAPI (see Energy poster for details)



SUPPORTED ARCHITECTURES

AMD	
ARM	Cortex A7, A8, A9, A15, X-Gene (ARM64), ARM1176 CPU (Original Raspberry Pi)
CRAY	
IBM	Blue Gene Series, EMON power on BG/Q; Power Series
Infiniband	
Intel	Nehalem, Westmere, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Knights Corner; RAPL, Power on Xeon Phi
NVidia	Tesla, Kepler, NVML, latest CUDA support for multiple GPUs



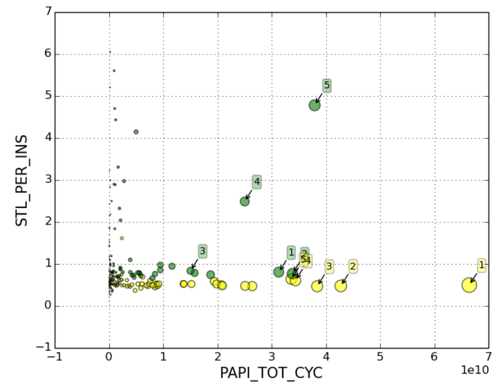
PAPI-NUMA

- Experimental (not yet released)
- Sampling support for cache and memory events, including data source, latency, etc.
- Intended to provide a standard interface to data needed for NUMA performance analysis and optimization

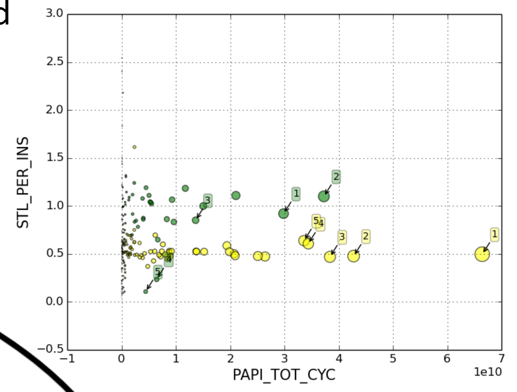
Autoperf

- *Simple* tool for performance experiments and associated analysis
- Adds a layer of abstraction over *existing* performance tools
- Automates tedious and error-prone tasks
  - Selecting performance counters (minimize # of experiments required)
  - Using available measurement tools: PAPI, TAU, HPCToolkit, Open|SpeedShop,...
  - Setting up the environment for each tool, managing sequential and batch parallel jobs on different architectures
  - Generating selective profiling configuration based on sampling results
  - Configuring access to databases (e.g. TAUdb), uploading data
  - Reusable and extensible analyses that are easy to understand; comparisons across multiple code versions
  - https://github.com/HPCL/autoperf.git
- Example: Studying the effects of optimizations on a Geant4 application (SimplifiedCalorimeter) compiled gcc 4.8 (any two versions can be compared this way).

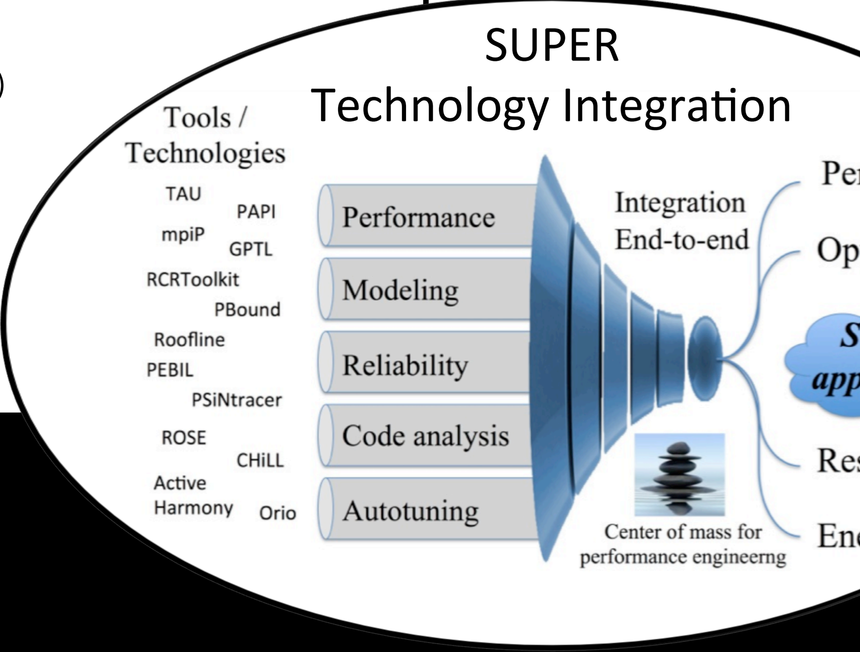
Stalls per instruction vs total cycles: -O2 unexpectedly increases stalls per instruction in two of the functions; each circle's diameter is proportional to the corresponding function's fraction of total execution time. The top 5 functions are labeled. Other measurements presented in a similar manner help determine the cause of the stalls.



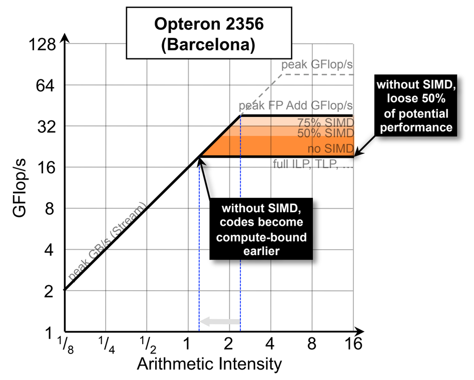
-O2 (green) vs. -O0 (yellow)



-O3 (green) vs. -O0 (yellow)



Empirical Roofline Toolkit



Background

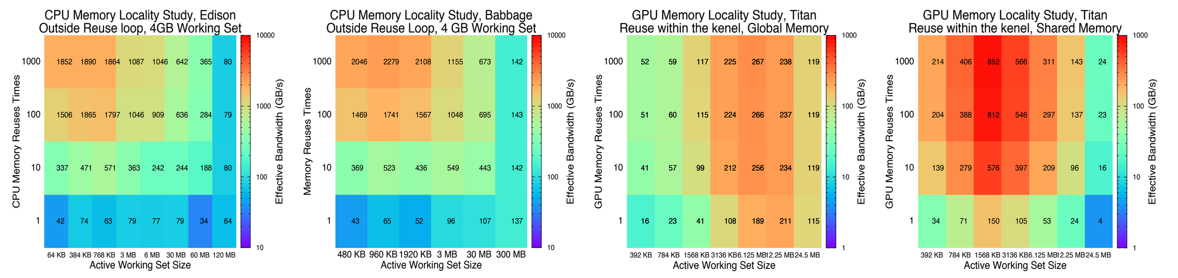
- The Roofline model provides a visually-intuitive approach to analyzing application performance.
  - Decomposes application into key numerical kernels
  - Principally oriented around throughput metrics (flop/s vs. GB/s)
  - Uses machine and application balance to determine performance bound
  - Expandable by including ILP, DLP, TLP, cache, and memory access pattern effects
- To date, application of the Roofline has been challenged on four fronts…
  - It requires a model of processor microarchitecture. Many researchers often lack the computer architecture background to create such a model.
  - It requires accurate monitoring of kernel execution involving DRAM data movement, SIMDization, ILP stalls, use of TLP, etc… This information is difficult to extract from some tools and impossible to gather from some processors.
  - It requires estimates of the data movement and computational requirements of each numerical kernel. e.g. what is the minimum data movement and computation each kernel requires? Within each kernel, is there any inherent DLP or ILP? Since existing tools are unable to extract these parameters, the model requires application scientists be knowledgeable of both computer architecture and application software (a rare combination).
  - Visualization of the model was left to the user. In practice, this varied from whiteboard doodles, to elaborate GNU and MATLAB plots.
- To that end, SUPER and FastMath have collaborated on developing a Roofline Toolkit to facilitate use of the model.

Initial ERT Release

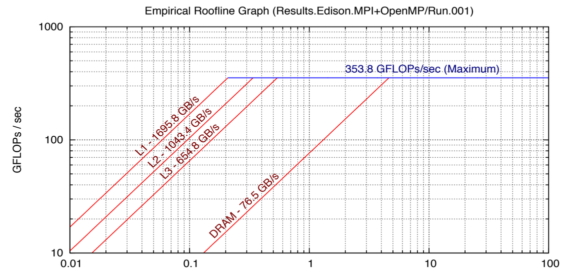
- Initial ERT release focused on characterizing and visualizing the Flop/DRAM Roofline on CPU architectures.
  - Peak flops (using polynomial amenable to FMA instructions)
  - Bandwidths and capacities for each level of memory and cache
- Runs on…
  - Xeon (Edison), Xeon Phi (Babbage), Opteron (Hopper), BlueGene/Q (Mira), Power7 and Power8
- Proxy real-world applications…
  - MPI+OpenMP implementation highlights any unintended NUMA issues.
  - Compiled C code (can the compiler SIMDize?)
  - Streaming (throughput-oriented) behavior with ample ILP, DLP, TLP
- Roofline Visualization…
  - Simple, portable Roofline chart viewer tool
  - Eclipse integration
  - Access to Roofline chart data stored in shared database

Beyond the Textbook Roofline Model

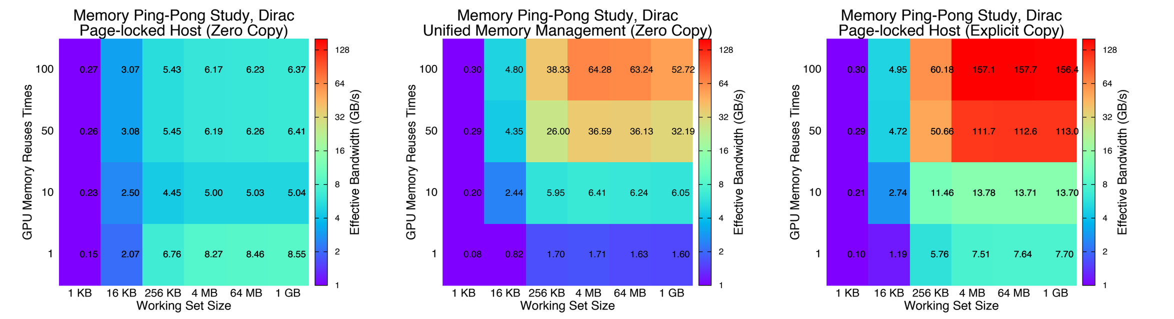
- Nominally, Roofline is a throughput-oriented (streaming) performance model on a single level of memory or cache.
- In reality, architectures have multiple levels of memory and applications have hierarchical working sets.
- Thus, reuse, bandwidth, and working set sizes are important metrics in understanding performance.
- Expanded Roofline to capture **performance on a two-level memory as a function of reuse and working set size…**
  - GPU performance is highly dependent on use of shared vs. cache (application writer must choose implementation on kernel by kernel basis).
  - CPUs are much faster than GPUs in some regions…



- CUDA 6.5 now supports **Unified Memory** (treat device memory as OS-controlled page cache on CPU memory)
- GPU Programmers must choose whether to…
  - use Unified memory and let the OS control everything
  - micromanage data allocation/movement/locality themselves
  - use zero copy memory and keep everything on the host.



- **How does performance vary on Kepler GPUs (e.g. ORNL's Titan and NERSC's Dirac)?**
  - Zero copy memory performs very poorly (PCIe bandwidth on every access) and has no benefit from temporal locality.
  - **Page locked with explicit management works well for large working sets (>4MB) with high temporal locality (reuse 50x).**
  - Unified memory behaves like explicit management getting a benefit from temporal locality, but is 3x slower.
  - It seems explicit management of data movement and locality is still required on Titan.



(Partial) support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics). LLNL-POST-657319.

OAK RIDGE National Laboratory | BERKELEY LAB | THE UNIVERSITY OF UTAH | UNIVERSITY OF OREGON | ICL INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE | UTEP

(Partial) support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/ Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/ Nuclear Physics) under award numbers DE-SC0006844, DESC0006947.