Leveraging Flash in HPC Systems

IEEE MSST

June 3, 2015

Brian Van Essen, CASC

Lawrence Livermore National Laboratory

LLNL-PRES-671909

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Opportunity

- HPC including node-local and near-line NVRAM
  - NVRAM bridges memory/storage latency gap

- Use cases:
  - Persistent data structures – reuse / save critical state
  - In-situ data staging – platform for coupling analysis with simulation
  - Extend memory – solve larger problems

- Use memory-mapping interface to access NVRAM
  - Exposes file as linear memory region
  - Simple load-store interface
  - Handle external data as internal data structures
  - Buffering and I/O are implicitly handled by O/S + runtime

Bring NVRAM into HPC

- Today's data-intensive HPC:
  - Catalyst
    - 150 teraflop/s: 324 nodes, 7776 cores
    - 40 TB DRAM + 253 TB Flash
    - Each node: 128 GiB DRAM + 800 GiB Flash

- Future NNSA's ASC HPC:
  - Sierra
    - 120-150 petaflop/s
    - 2.1-2.7 PB DRAM + 3.3-4.2 PB Flash
    - Each node: 512GiB DRAM+HBM & 800 GiB NVMe Flash

Motivation

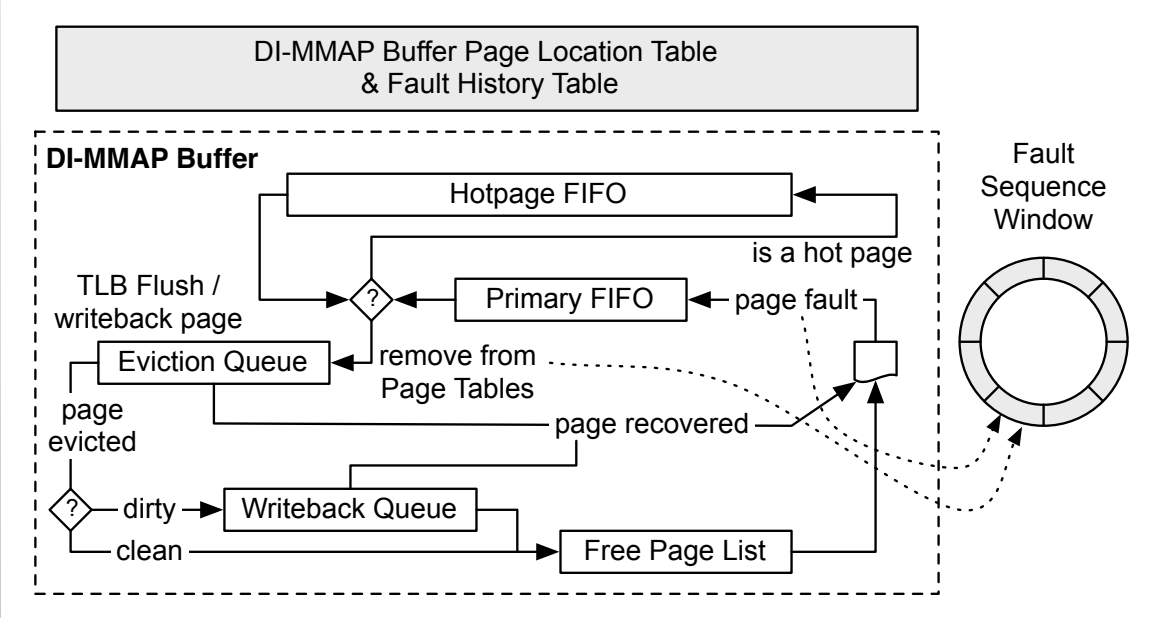
- Enable scalable out-of-core computations for data-intensive computing.
- Effectively integrate non-volatile random access memory into the HPC node's memory architecture.
- Address data-intensive computing scalability challenges:
  - Use node-local NVRAM to support larger working sets
  - DRAM-cached NVRAM to extend main memory
- Allow latency-tolerant applications to be oblivious to transitions from dynamic to persistent memory when accessing out-of-core data.

Tuning memory-mapped I/O for data-intensive applications

- Linux memory map runtime:
  - Optimized for shared libraries
  - Does not expect memory-mapped data to churn
  - Does not expect memory-mapped data to exceed memory capacity

- Data-Intensive Memory-Map runtime (DI-MMAP):
  - Optimized for frequent evictions
  - Optimized for highly concurrent access
  - Expects to churn memory-mapped data

Advanced buffer management and caching techniques



Minimize the amount of effort needed to find a page to evict:

- In the steady state a page is evicted on each page fault
- Track recently evicted pages to maintain temporal reuse
- Allow bulk TLB operations to reduce inter-processor interrupts

Experience with data-intensive applications

- Large-scale graph analysis
- Metagenomic analysis
- Streamline tracing

- Characteristics of data-intensive applications:
  - Large data sets
  - Large working sets that exceed capacity of main memory
  - Memory bound
    - irregular data access
    - latency sensitive
    - minimal computation
- Methods of tuning applications for NVRAM (adding latency-tolerance):
  - Concurrent I/O
  - Avoid bulk synchronous communication
  - Potentially asynchronous execution

Graph Analytics (BFS Graph500)

- HavoqGT Graph Library
  - BFS, SSSP, Connected components, K-core, PageRank
  - Level-asynchronous
  - Delegate-partitioning
  - Highly concurrent
    - Multi-process and multi-thread implementations

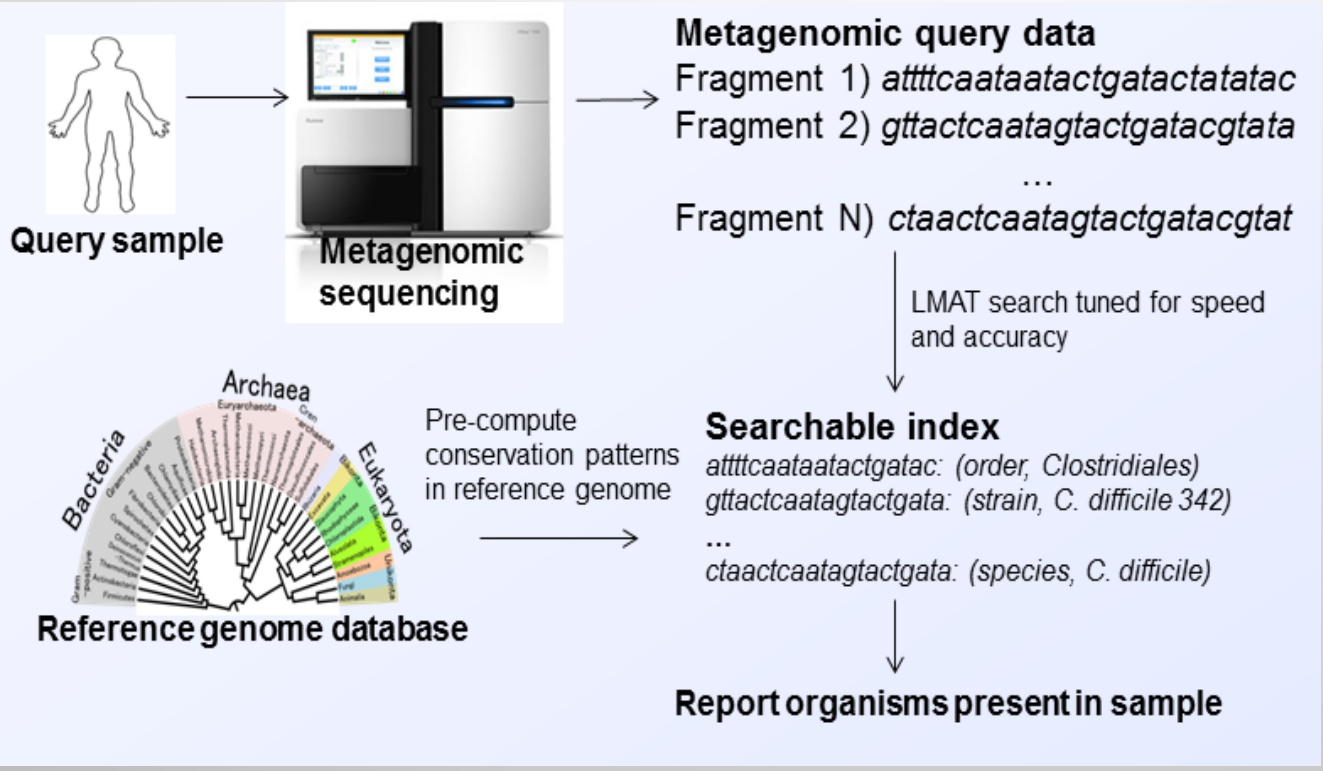
Solving larger problems:

- Distributed memory (multi-process) (300 nodes)
  - In-memory (DRAM Only): Scale 36
  - External-memory (DRAM+NVRAM): Scale 40
- Single node
  - In-memory (DRAM Only): Scale 28
  - External-memory (DRAM+NVRAM): Scale 32

Graph500 #4 – Distributed node-local

- HavoqGT Graph Library – BFS
- Tied for 2nd in size and placed #4 in Nov. 2014
  - when ranked by size (second only to full Sequoia BG/Q)
- Scale 40 graph (17.6 trillion edges)
  - 217 TB
- Catalyst cluster – 300 nodes
  - 24 cores per node
  - 128GB DRAM + 800 GB NVRAM per node
  - 24 processes per node
- DI-MMAP runtime
  - 2.4x improvement over Linux mmap

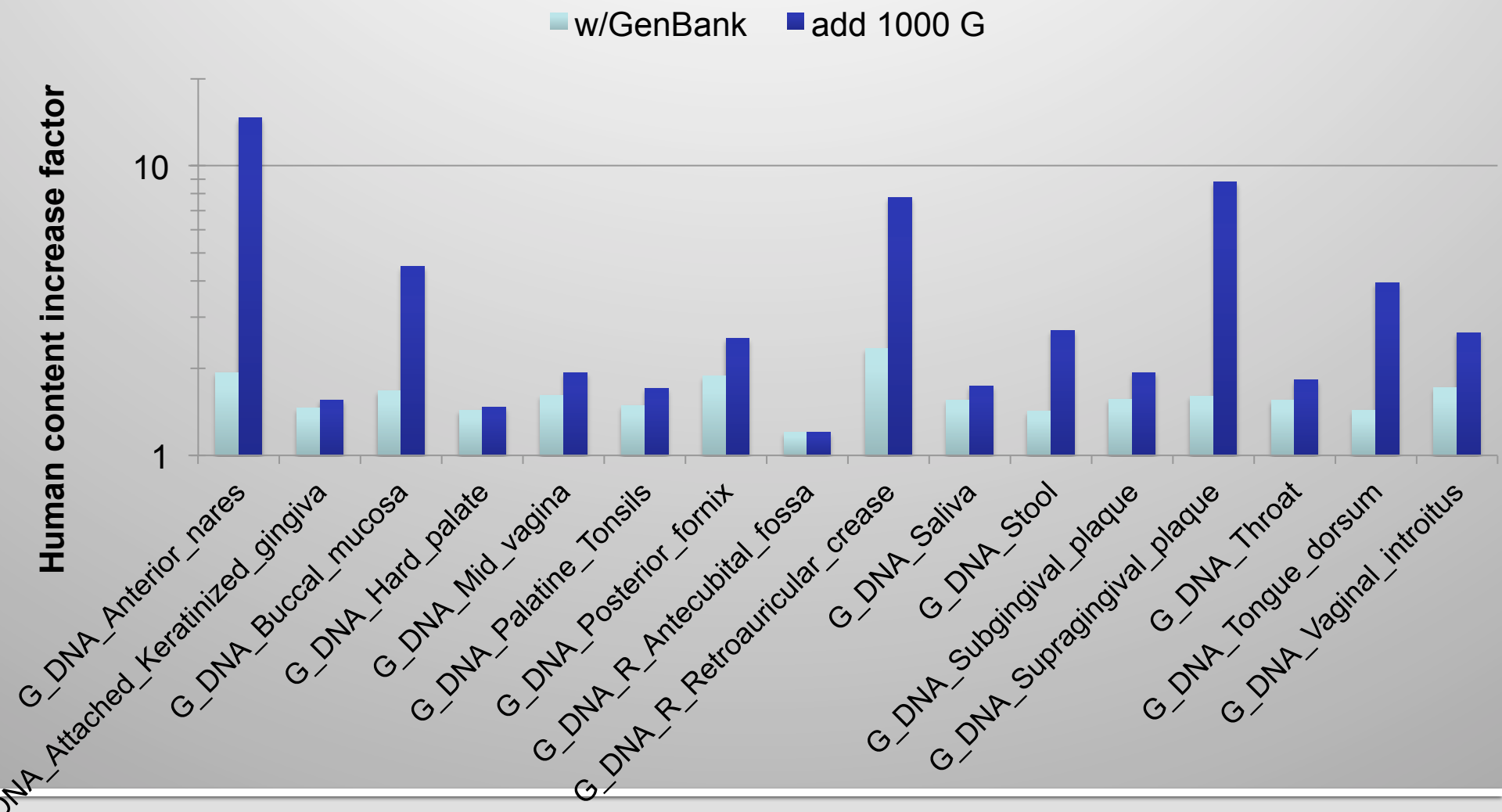
Metagenomic analysis conducted on an unprecedented scale using NVRAM and Catalyst



Search index is too large to fit in traditional memory so stored on the flash drive of each Catalyst node and accessed as if in memory. DI-MMAP provides 3-4x speedup.

Large scale analysis	Size	Run time	Result
1000 Genome Project (2,646 people)	90 Terabases	6 days	Identified 8 million new genetic variants
Human Microbiome (9,113 samples)	18 Terabases	38 hours	New human sequence and microbial species

Larger, more accurate DB means better classification



Next generation runtime interfaces

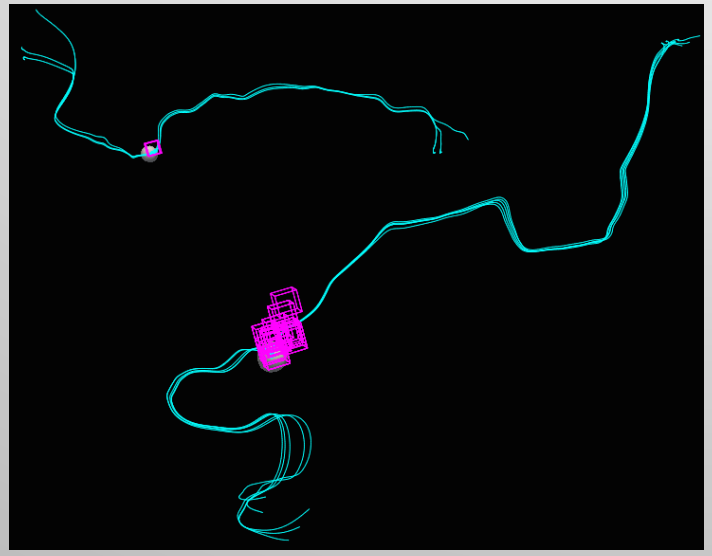
- Go beyond coarse system-level tuning
  - Per-application or per-data structure tuning
  - Data-dependent decisions

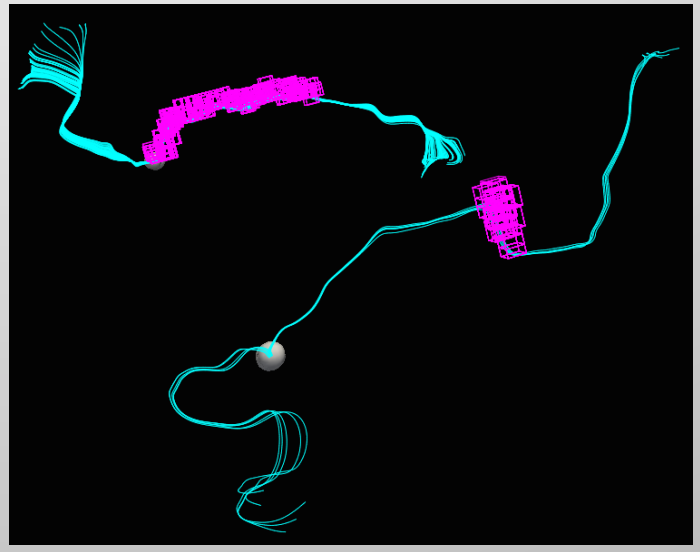
- Advanced features
  - Variable sized I/O requests (runtime-level superpages)
    - allow I/O at non-native page size
    - optimize I/O transfer to maximize bus utilization
    - tune per-buffer or data-structure
  - Lightweight buffer introspection
    - non-faulting page residency check (data-dependent scheduling)
    - page NUMA-node location (spatial scheduling)
    - page fault frequency (identify hot or cold pages)

Mapping metrics back to application space

- Visualize (overlay) active buffer pages on application data structures
  - Streamlines are shown from seed point (silver sphere) to termination
  - Pink cubes show the active pages in the buffer for the current time step

Time Step 307





Time Step 327

- Spatial distribution of buffered pages w.r.t. streamlines illustrate reuse within streamline clusters
  - Identifiers potential opportunity for intelligent pre-fetching
- Fully tracing long streamlines serially leads to less data reuse between seeds

Managing NVRAM with HPC job scheduling

State of the practice:

- Job allocation on stateless nodes
  - Primary concern is node-allocation within routing topology

Challenges:

- Is my data in the system
- How is my data laid out
  - Does the data layout match the MPI rank layout
- Increased read activity to parallel file systems

Managing NVRAM with HPC job scheduling

Current policy:

- Data-retention
  - Leave on node (allows reuse)
  - Flush on demand (job allocation)
  - No guarantee from allocation to allocation
- Security
  - Standard Linux user permissions

Future work:

- Scheduling NVRAM resources
  - Guarantee that there is sufficient capacity
  - Re-allocating nodes that your data (in the correct distribution)
- Building global, distributed store
  - State between HPC jobs

Conclusions: Leveraging NVRAM in HPC

- With the right system software stack and algorithms:
  - NVRAM can be an effective tier in the memory hierarchy
  - Invisible to the application programmer as a separate resource.
- Solve larger problems
- Reduces the amount of main memory needed (thus reducing power)
  - provides capacity
  - unused capacity does not require dynamic power
- It requires a fresh look algorithm design:
  - minimize latency sensitivity
  - transform to be throughout driven
- I/O performance scales well with ratio of buffer size to problem size
- Optimized caching in runtime can minimize NVRAM perceived latency, wear, and power


Lawrence Livermore
National Laboratory