Extreme Heterogeneity in Emerging Memory Systems

Jeffrey S. Vetter

With many contributions from FTG Group and Colleagues

MCHPC Panel
SC19
18 Nov 2019

http://ft.ornl.gov vetter@computer.org

NVM Design Choices

Architectural considerations

- Exploit persistence
  - ACID?
- Integration point
  - Memory
  - Node
  - System
- Scalability
- Programming model

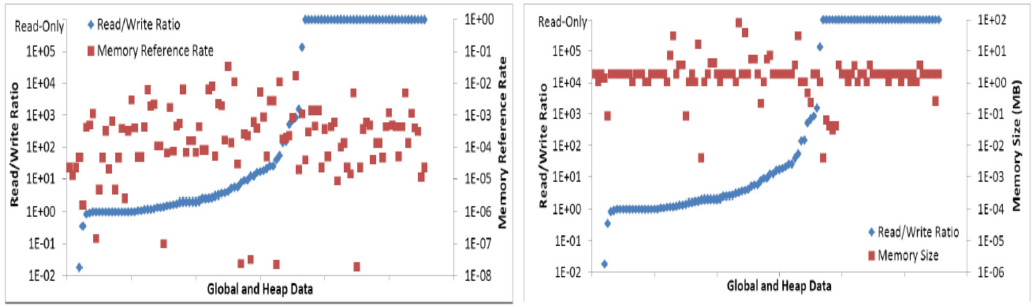
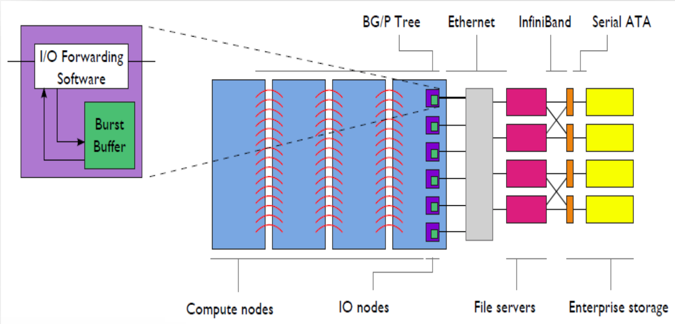


Figure 3: Read/write ratios, memory reference rates and memory object sizes for memory objects in Nek5000

Application Scenarios

- Burst buffers
- In situ viz and analytics
- Persistent data structures

[Liu, et al., MSST 2012]

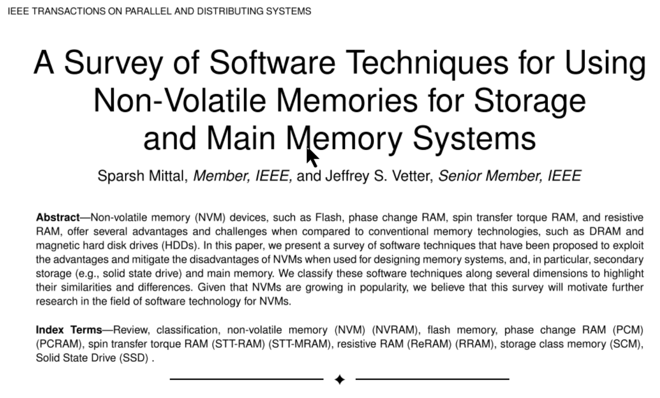


Empirical results show many reasons…

- Lookup, index, and permutation tables
- Inverted and 'element-lagged' mass matrices
- Geometry arrays for grids
- Thermal conductivity for soils
- Strain and conductivity rates
- Boundary condition data
- Constants for transforms, interpolation
- MC Tally tables, cross-section materials tables…

Our Approaches

- Transparent access to NVM from GPU
- NVL-C: expose NVM to user/applications
- Papyrus: parallel aaggregate persistent memory
- Many others (See S. Mittal and J. S. Vetter, "A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems," in IEEE TPDS 27:5, pp. 1537-1550, 2016)



IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTING SYSTEMS

A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems

Sparsh Mittal, *Member, IEEE*, and Jeffrey S. Vetter, *Senior Member, IEEE*

Abstract—Non-volatile memory (NVM) devices, such as Flash, phase change RAM, spin transfer torque RAM, and resistive RAM, offer several advantages and challenges when compared to conventional memory technologies, such as DRAM and magnetic hard disk drives (HDDs). In this paper, we present a survey of software techniques that have been proposed to exploit the advantages and mitigate the disadvantages of NVMs when used for designing memory systems, and, in particular, secondary storage (e.g., solid state drive) and main memory. We classify these software techniques along several dimensions to highlight their similarities and differences. Given that NVMs are growing in popularity, we believe that this survey will motivate further research in the field of software technology for NVMs.

Index Terms—Review, classification, non-volatile memory (NVM) (NVRAM), flash memory, phase change RAM (PCM) (PCRAM), spin transfer torque RAM (STT-RAM) (STT-MRAM), resistive RAM (ReRAM) (RRAM), storage class memory (SCM), Solid State Drive (SSD) .

http://j.mp/nvm-sw-survey

DRAGON : Expanding the memory capacity of GPUs

- GPUs have limited memory capacity
- Recent GPUs have added paging support to host memory
- Recent datasets have grown larger that host memory
- Extend GPUs to NVM
  - Support for massive data
  - Support for temporary data
  - Support for read-only data
- Good performance (including surprises)

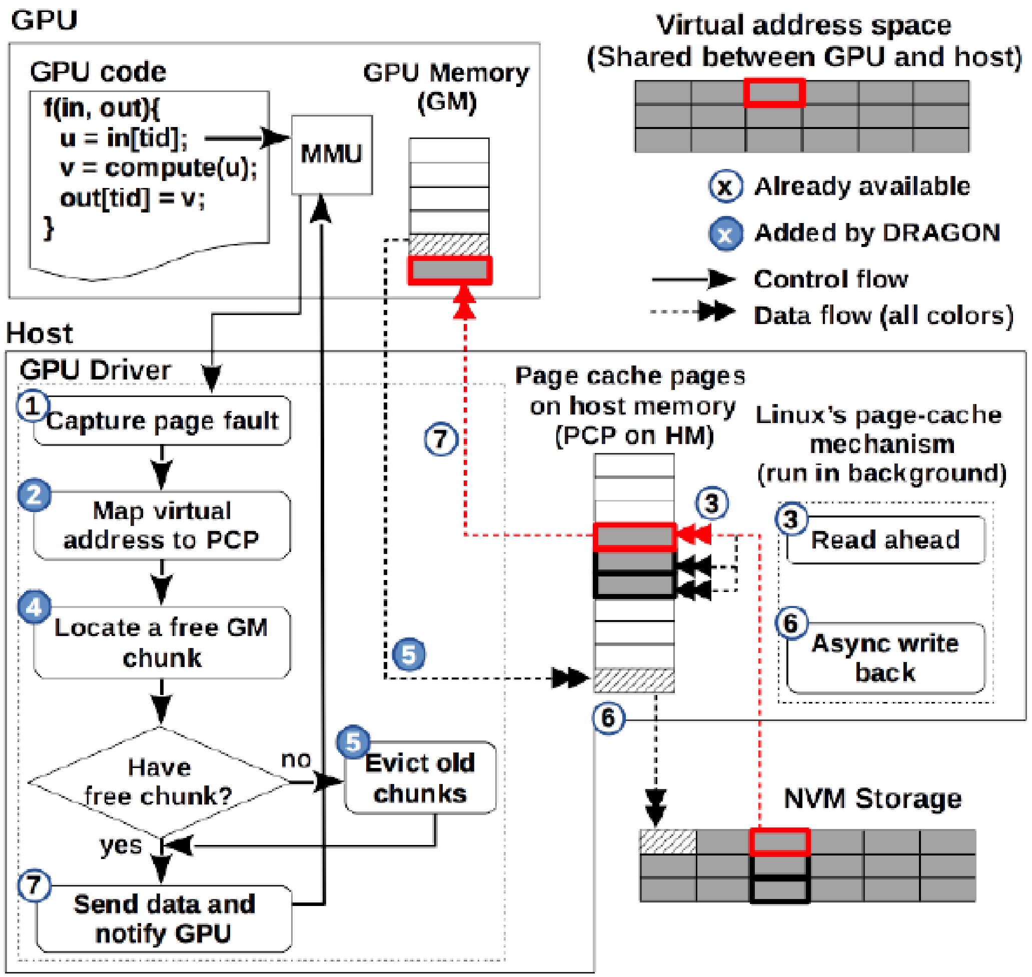


Fig. 1: *DRAGON* driver operation

P. Markthub, M.E. Belviranli *et al., "DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access," in SC18, 2018*   https://github.com/pakmarkthub/dragon

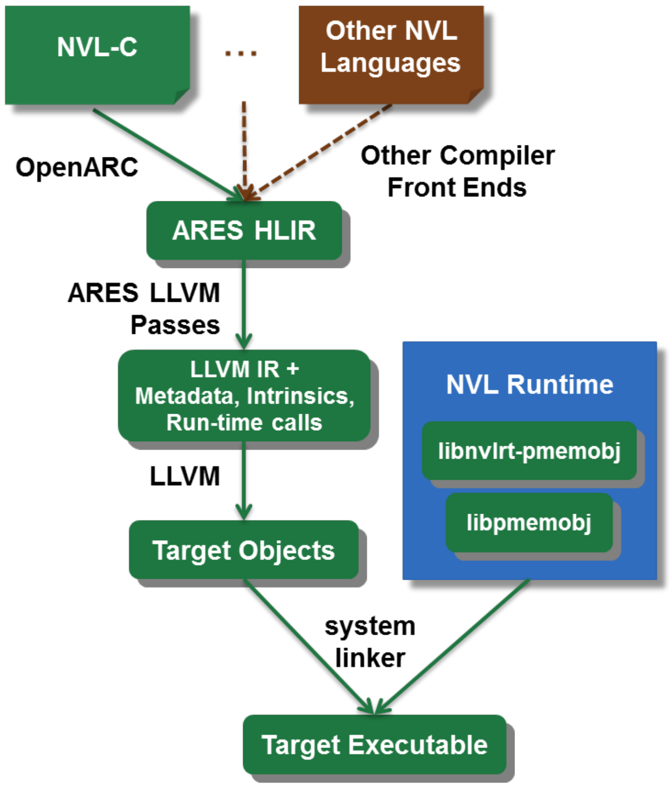
NVL-C: Portable Programming for NVMM

- Minimal, familiar, programming interface:
  - Minimal C language extensions.
  - App can still use DRAM.
- Pointer safety:
  - Persistence creates new categories of pointer bugs.
  - Best to enforce pointer safety constraints at compile time rather than run time.
- Transactions:
  - Prevent corruption of persistent memory in case of application or system failure.
- Language extensions enable:
  - Compile-time safety constraints.
  - NVM-related compiler analyses and optimizations.
- LLVM-based:
  - Core of compiler can be reused for other front ends and languages.
  - Can take advantage of LLVM ecosystem.

```c
#include <nvl.h>
struct list {
  int value;
  nvl struct list *next;
};
void remove(int k) {
  nvl_heap_t *heap
    = nvl_open("foo.nvl");
  nvl struct list *a
    = nvl_get_root(heap, struct list);
  #pragma nvl atomic
  while (a->next != NULL) {
    if (a->next->value == k)
      a->next = a->next->next;
    else
      a = a->next;
  }
  nvl_close(heap);
}
```

Pointer Class	Permitted
NV-to-V	no
V-to-NV	yes
intra-heap NV-to-NV	yes
inter-heap NV-to-NV	no

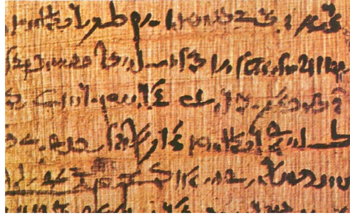
Table 1: Pointer Classes



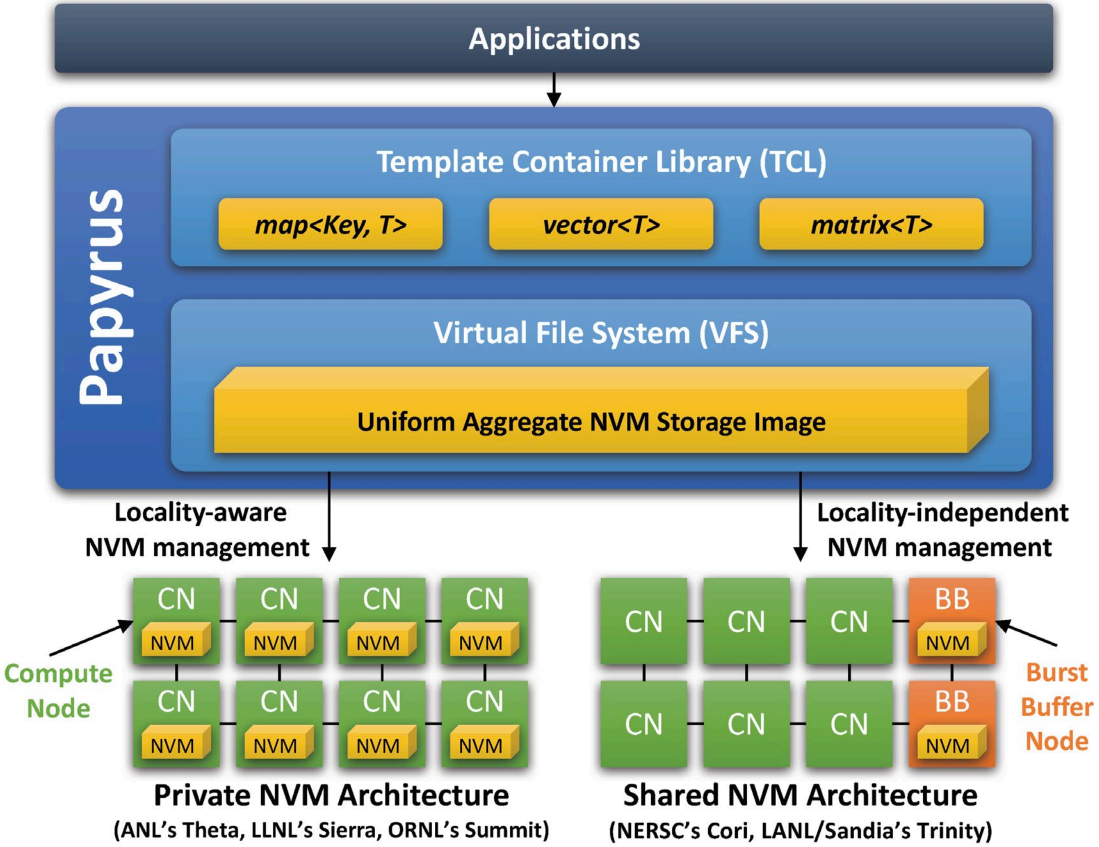
J. Denny, S. Lee, and J.S. Vetter, "NVL-C: Static Analysis Techniques for Efficient, Correct Programming of Non-Volatile Main Memory Systems," in *ACM High Performance Distributed Computing (HPDC). Kyoto: ACM, 2016*

Papyrus

**Wikipedia: Papyrus can refer to a document written on sheets of papyrus, an early form of a book.*



- ***Papyrus*** is a novel programming system for aggregate NVM in the next generation HPC systems
- Leverage emerging NVM technologies
  - High performance
  - High capacity
  - Persistence property
- Designed for the next-generation DOE systems
  - Portable across local NVM and dedicated NVM architectures
  - An embedded distributed key-value store (no system-level daemons and servers)
  - Scalability and performance
- Designed for HPC applications
  - MPI/UPC-interoperable
  - Application customizability
    - Memory consistency models (sequential and relaxed)
    - Protection attributes (read-only, write-only, read-write)
    - Load balancing
  - Zero-copy workflow, asynchronous checkpoint/restart



https://code.ornl.gov/eck/papyrus

[1] J. Kim, S. Lee, and J.S. Vetter, "PapyrusKV: a high-performance parallel key-value store for distributed NVM architectures," in *SC17.*

[2] J. Kim, K. Sajjapongse, S. Lee, and J.S. Vetter, "Design and Implementation of Papyrus: Parallel Aggregate Persistent Storage," in IPDPS 2017.