Extreme Heterogeneity in Emerging Memory Systems

Jeffrey S. Vetter

With many contributions from FTG Group and Colleagues

MCHPC Panel
SC19
18 Nov 2019

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

http://ft.ornl.gov  vetter@computer.org
NVM Design Choices

Architectural considerations

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

Application Scenarios

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

[Liu, et al., MSST 2012]

Our Approaches

- Transparent access to NVM from GPU
- NVL-C: expose NVM to user/applications
- Papyrus: parallel aggregate persistent memory

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

Figure 3: Read/write ratios, memory reference rates and memory object sizes for memory objects in Nck5000

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 than host memory
- Extend GPUs to NVM
  - Support for massive data
  - Support for temporary data
  - Support for read-only data
- Good performance (including surprises)

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);
}
```

Table 1: Pointer Classes

<table>
<thead>
<tr>
<th>Pointer Class</th>
<th>Permitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV-to-V</td>
<td>no</td>
</tr>
<tr>
<td>V-to-NV</td>
<td>yes</td>
</tr>
<tr>
<td>intra-heap NV-to-NV</td>
<td>yes</td>
</tr>
<tr>
<td>inter-heap NV-to-NV</td>
<td>no</td>
</tr>
</tbody>
</table>

Papyrus

- **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

---

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


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