Towards Optimizing Memory Mapping of Persistent Memory in UMap

Karim Youssef [1,2], Keita Iwabuchi [2], Wu-chun Feng[1], Maya Gokhale[2], Roger Pearce[2]

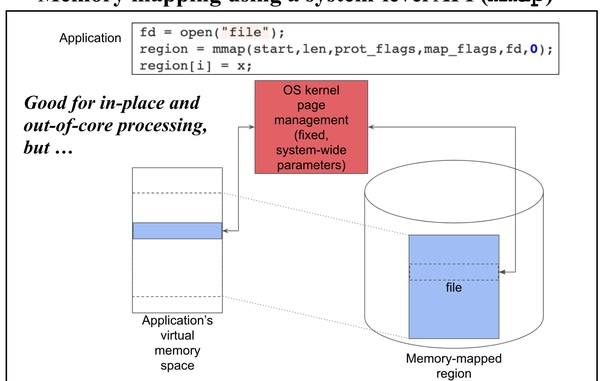
[1]Virginia Tech, [2]Lawrence Livermore National Laboratory

Introduction

- **Trends**
  - Dataset sizes? *Growing exponentially in many domains.*
  - Storage technologies? *Providing efficient data storage and processing capabilities*, e.g., byte-addressable and block-addressable persistent memory.
  - Abstract interfaces? *Enabling memory mapping of files on different storage types via interface to applications.*
- **Problem**
  - I/O contention on mapped files.
  - Storage inefficiency for sparse data structures.
- **Our Solution:** *Transparently partitioning a memory-mapped persistent region into multiple files with a sparse and dynamic allocation strategy.*

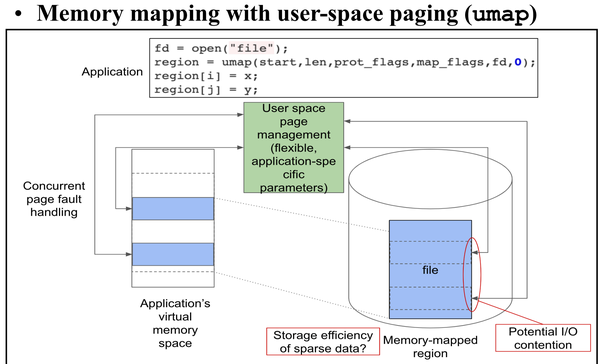
Background

- **Memory mapping using a system-level API (`mmap`)**



System-level page management lacks the flexibility for application-specific optimizations.

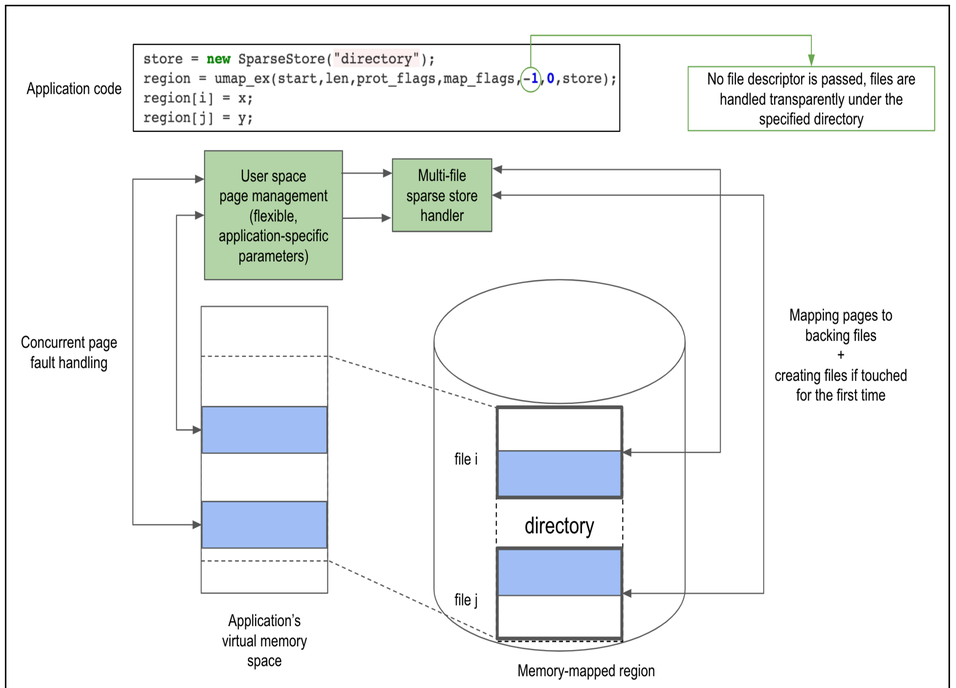
- **Memory mapping with user-space paging (`umap`)**



User-space page management offers flexible configuration of parameters (e.g., page size and concurrency) via user-space page management.

Approach

- Memory mapping using UMap and *SparseStore*, a multi-file backing store handler that uses a sparse allocation strategy to create files dynamically and as needed.
- Features
  - Multi-file → *Improves* parallel I/O performance by reducing I/O contention.
  - Dynamic on-demand creation of files → *Optimizes* the storage of sparse data structures.
  - Extension of the default handler → *Realizes* the functionality of the *SparseStore* handler.

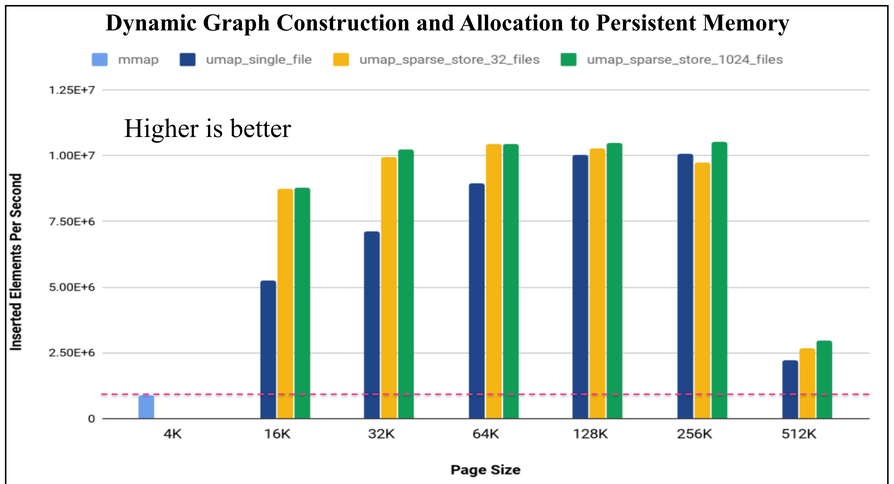


Memory mapping using Umap and *SparseStore* handler. The memory-mapped region is logically partitioned into multiple files that are created dynamically when pages are written for the first time. Metadata such as file granularity is saved to enable re-mapping of a region created using the *SparseStore* handler.

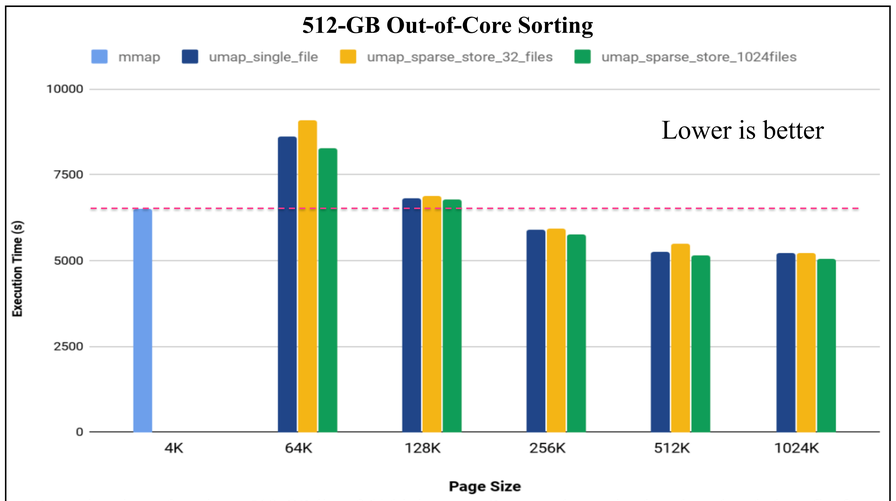
- No actual file is mapped at the beginning. *SparseStore* initially creates an empty directory.
- The *SparseStore* handler dynamically creates files with a configurable file granularity.
- When a page fault is serviced, the *SparseStore* handler maps the page's starting address to a file index and a file offset. The file gets created if it is touched for the first time.
- Metadata such as file granularity is saved to a file to enable subsequent remapping of a region that was created using *SparseStore* .

Experiments and Results

- Application Workloads: (1) Dynamic graph construction and (2) out-of-core sorting.
- Experimental Setup
  - Dual 24-core AMD EPYC 7401 CPUs
  - 256-GB DDR4 DRAM
  - 1.8-TB local NVMe SSD



Throughput of constructing a graph with size ~256GB and allocating as an edge list on persistent memory using 48 threads. Using UMap with *SparseStore* and 1024 backing files yielded ~12x speedup over mmap at 32K page size and ~2x speedup over UMap with a single mapped file at 16K page size.



Execution time of sorting a 512-GB list of 64-bit integers. Because the sorting benchmark only benefits from UMap at larger page sizes, increasing the number of backing files did *not* deliver significant improvement.

Conclusion

- Using multiple backing files that are created dynamically benefits the performance of applications with highly irregular memory access patterns at smaller page sizes (up to 64 KB).
- Using UMap with the *SparseStore* handler delivers up to 12 times faster performance than system-level *mmap* and up to 2 times faster than UMap with the default store handler that maps a single file.
- The *SparseStore* implementation is available as a part of UMap: https://github.com/LLNL/umap

Prepared by LLNL under contract DE-AC52-07NA27344 (LLNL-POS-813825)
Experiments were performed at the Lawrence Livermore Computing facility.