Motivation

- Improve I/O efficiency
- Maintain simple interface to NVRAM
  - Memory-mapped I/O
- Increase user control
- Extended capabilities of memory-map interface
  - Variable sized I/O transfers
  - Decouple memory-map fault from system page size
- Target data-intensive applications
  - Streamline tracing, large scale graph analysis, metagenomics
- Leverage data-intensive memory-map runtime (DI-MMAP) research platform
  - Create superpage construct within DI-MMAP buffer
Optimizing data transfer size

- Large I/O requests
  - Higher bandwidth
  - Better bus utilization
  - Shorter average latency per byte
  - Longer latency per request
  - Increased cost for over-fetching data
  - Increased capacity conflicts

- Small I/O requests
  - Lowest latency per request
  - Longer average latency per byte
  - Increased number of requests
Memory-mapped I/O

- Traditionally integrally tied to system page size
  - Huge pages are good for anonymous memory and in-memory heap allocation

- x86-64 page sizes: 4KB, 2MB, 1GB

<table>
<thead>
<tr>
<th>Small Pages (4KB)</th>
<th>Large/Huge Pages (2MB, 1GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained data movement</td>
<td>High spatial locality</td>
</tr>
<tr>
<td>Quick fault resolution</td>
<td>Long fault resolution</td>
</tr>
<tr>
<td>Poor TLB coverage</td>
<td>Good TLB coverage</td>
</tr>
<tr>
<td>Lots of book keeping</td>
<td>Minimized PTE overhead</td>
</tr>
<tr>
<td>Less efficient I/O transfer</td>
<td>Increased cache capacity conflicts</td>
</tr>
</tbody>
</table>

- Data-intensive applications
  - Irregular data access patterns
  - out-of-core data structures

... huge pages are not a solution
Tuning memory-mapped I/O

- Extend custom data-intensive memory-mapped I/O runtime
- Create “super”-page that is composed of multiple system pages
  - superpage sizes are $2^N$ system pages
- Each page fault brings in a superpage-aligned chunk of data
- DI-MMAP extensions:
  - allocate buffer pages in contiguous chunks
  - user process memory-maps physical buffer pages to create a “bounce” buffer
  - hack direct I/O from the kernel to target the “bounce” buffer
What is the DI-MMAP runtime?

A high-performance alternative to Linux mmap:

- performance scales with increased concurrency
  - both multi-threading and multi-process
- performance does not degrade under memory pressure
- associate data structures with distinct di-mmap buffers

DI-MMAP features:

- loadable kernel module
- a fixed sized page buffer (independent of Linux page cache)
- minimal dynamic memory allocation
- efficient FIFO buffer replacement policy
- preferential caching for frequently accessed pages
- explicit control for page fault size (via superpages)
Linux & DI-MMAP Basic Design

- 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

- DI-MMAP:
  - Optimized for frequent evictions
  - Optimized for highly concurrent access
  - Expects to churn memory-mapped data
DI-MMAP buffer management techniques for NVRAM pages

- Physical page array is memory mapped into user application address space
- Mapped array provides an addressing trick that allows the kernel to perform direct I/O into the physical page array
- Location hashtable and fifo track superpages rather than system pages
Superpage fault handling: 16K page

**Superpage Fault**

- Amortize I/O request and response overhead
- Subsequent accesses wait for fault to complete
- Additional pages are wired into PTE
  - Avoid further faults
Streamline tracing is an important tool for visualizing and analyzing flow fields.

- Computing parallel streamlines out-of-core efficiently is difficult
  - Data set size
  - Seeding density and distribution
  - Flow field complexity

- Underlying cause: \textit{irregular} and \textit{data-dependent} access patterns
Raleigh-Taylor Simulation

Velocity magnitude

Feature-based seeded streamlines
Prior work fine-tuning streamline tracing with implicit and explicit I/O

- Memory-mapped I/O
  - Very good performance
  - Best with high concurrency

- User-level buffer
  - Variable I/O request size
  - Peak performance at 192KB

- What about tuned memory-map I/O
  - led to development of superpage I/O
Streamline Tracing: Hilbert Layout & row-major seeding

- Minimizing page faults does not maximize performance: 23.7% @ 64KB
- Peak performance with 16KB pages: 47.2% faster than 4KB pages
Next steps

- Test efficacy of superpages for large scale graph analysis with HavoqGT library
  - Optimized for 16KB access patterns

- Support multiple page sizes within DI-MMAP buffer
  - Allow per data structure superpage size
    - How does the user indicate superpage size for each data structure… madvise per VMA
    - Manage pools of multiple sized allocations… can a set of smaller pages be substituted in a region marked for larger pages
Conclusions: Superpages improve performance and runtime flexibility

- Larger I/O requests are efficient (if most of the data is used)
  - Better bus utilization
  - Amortized request / response
  - Natural prefetching

- Over-fetching data increases cache contention
  - Especially for data-intensive applications with irregular access patterns

- Huge pages are great for heap allocation and in-memory data structures
  - Too much superfluous data-movement for data-intensive

- Customized data-intensive memory-map runtime (DI-MMAP)
  - Decouples fault size from system page size
  - Optimizes data movement for each memory-map buffer
  - Improves memory-map performance by 40-53% for streamline tracing
Streamline Tracing: Block row-major layout & seeding

- Minimizing page faults does not maximize performance: 32.0% @ 64KB
- Peak performance with 32KB pages: 40.8% faster than 4KB pages
Streamline Tracing: Block row-major layout & random seeding

- Minimizing page faults does not maximize performance: 48.7% @ 64KB
- Peak performance with 16KB pages: 53.6% faster than 4KB pages
Streamline Tracing: Hilbert layout & random seeding

- Minimizing page faults does not maximize performance: 33.4% @ 64KB
- Peak performance with 16KB pages: 50.8% faster than 4KB pages