Data Pipes
Declarative Control over Data Movement

Lukas Vogel, Daniel Ritter, Danica Porobic, Pınar Tözün, Tianzheng Wang, Alberto Lerner
Status Quo: Data movement is a bottleneck

- Wildly different hardware at every level
  - Characteristics (latency, throughput, …)
  - Granularity (Cache line, page, block, …)
  - Access patterns (sequential, random)

- Applications spend a lot of time moving data
  - Explicitly: e.g., pread
  - Implicitly: e.g., CPU caches

- Moving data is complex
  - APIs (pread, mmap, io_uring, SPDK, S3,…)
  - Protocols (NVMe/PCI, HTTP, SATA, DDR(-T))
  - Instructions (CLFLUSH(OPT), CLWB, CLDEMOTE)
  - CPU involvement

- We’d like to utilize “shortcuts”
  - Such as DMA (I/OAT), DDIO
  - Exist between nearly any device pair
  - We call them “data movement primitives”
Reason 1: Primitives are hard to control

- DDIO reduces data traffic
- Same command, but two paths
- Only tuning knob: global undocumented MSR

⇒ Hard to control!
Reason 2: Primitives are hard to target

- Same sink, but two paths
- Fast path hard to develop for

⇒ Hard to use!
Primitives have to be re-thought

● Problem: Hard to utilize, hard to control
  ○ Vendor-specific, transparent
  ○ Varying APIs, unconventional or not widely supported

● The culprit: implicit, bottom-up approach
  ○ Vendors see primitives as accelerators for specific use-cases
  ○ Breaks down when straying from the happy path

● Proposal: Declarative, top-down data movement
  ○ Build a unified abstraction
  ○ Give control to the developers
  ○ We call it: Data Pipes
Data Pipes

- “Typed” locators make **data location** explicit
- Pipes make **data movement** explicit
- **Top-down** approach is **declarative** and shows **intentions**
- Framework underneath decides on how data is actually moved
  - Employ DMA hardware capabilities
  - Software fallback
Data Pipes Example

SSDResourceLocator ssd("file/path");
DRAMResourceLocator dram(1 * GB);

parallel_per_core {
  CacheResourceLocator cache(CACHE_SZ);
  Pipe ssd_up(ssd, cache);
  Pipe cache_down(cache, dram);
  ssd_up.transfer(1 * MB);
  sort(cache, 1 * MB);
  cache_down.transfer(1 * MB);
}

Please take a look at the paper for an inversion of control and an OS-supported approach.
Properties of Data Pipes (1/2)

- Declarative
- Composable
- Optimizable
Properties of Data Pipes (2/2)

- Configurable
- Orthogonal to existing primitives
- Adaptable to new primitives

- clflush, cldemote
- SPDK, NVMe, DDIO
- I/OAT, Prefetch, ...
Research Agenda

● Current Prototype: Data Pipes in user space (e.g., SPDK)

● Integrate Data Pipes into the OS
  ○ Common baseline for different libraries
  ○ Reserve resources managed by the Data Pipe runtime (e.g., no swapping)
  ○ API with epoll: See paper!

● Cloud Infrastructure Opportunities
  ○ Predictability => Reduce overprovisioning (noisy neighbors!)
  ○ Common API helps customers to migrate to new device generations
Hardware Vendor Wishlist

- **Give us more access to the hardware!**
  - Expose state and tuning knobs of DDIO, I/OAT, …
  - More control over the caching layer

- **Intel’s integrated DMA unit lacks some features**
  - Lacking documentation
  - Low bandwidth for smaller data transfers
  - Limited number of channels

- **Increased reach of DMA units**
  - Some device pairs have no acceleration, e.g. DMA directly into the registers?
Conclusions

- Data movement is a bottleneck in data-intensive systems
- Existing data movement primitives are heterogeneous and hard to use
- **Data Pipes**: Abstraction to …
  - … present a uniform interface to the application
  - … make data movement primitives
    - top-down,
    - declarative,
    - and explicit.

Thank you for your attention!
Inversion of Control approach

SSDResourceLocator ssd("file/path");

DRAMResourceLocator dram(1*GB);

PipeRuntime runtime;

runtime.fork_and_start();

vector<future> futures;

Pipe dram_down(dram, ssd, &runtime);

parallel_per_core {
    promise<void> write_promise;
    futures.push_back(write_promise.get_future());
    // Do some computation with data in DRAM
    dram_down.transfer(20 * MB, write_promise);
}

wait_all(futures).wait();
futures.clear();
OS supported approach

SSDResourceLocator ssd(“file/path”);

DRAMResourceLocator dram(1*GB);

int dram_downpipe_fd = create_pipe(dram, ssd);

int epoll_fd = epoll_create1(0);

epoll_event dram_pipe_op;

dram_pip_op.events = EPOLLTRANSFER;

dram_pip_op.fd = dram_downpipe_fd;

epoll_ctl(epoll_fd, EPOLL_CTL_ADD, 0, &dram_pipe_op)

parallel_per_core {

// Wait until the pipe has capacity to transfer

epoll_event event;

epoll_wait(epoll_fd, &event, 1);

pipe_transfer(dram_downpipe_fd);

}