Memory-Driven Computing

Kimberly Keeton
Distinguished Technologist
kimberly.keeton@hpe.com

October 20, 2016
From Processor-Centric Computing…

…to Memory-Driven Computing
Technology trends enabling Memory-Driven Computing

- Converging memory and storage
  - Byte-addressable persistent memory (NVM) replaces hard drives and SSDs

- Resource disaggregation leads to shared memory pool
  - Fabric-attached memory pool is accessible by all compute resources
  - Optical networking provides near-uniform latency
  - Local volatile memory provides lower latency, high performance tier

- Distributed heterogeneous compute resources
  - Moving compute closer to data

- Software
  - Memory-speed persistence
  - Low-latency, high BW memory/storage access
  - Globally addressable, low latency access to all PM across the memory fabric
Memory-Driven Computing in context

Shared nothing

Shared something

Shared everything
Outline

– Overview
– Memory driven systems software: OS and data management
– Memory driven data analytics: Spark for The Machine
– Memory driven programming models
– Prototypes and emulators
– Commercial memory driven computing
– Summary
Memory driven systems software
Traditional file systems

- Separate storage address space
  - Data is copied between storage and DRAM
  - Block-level abstraction leads to inefficiencies
- Use of page cache leads to extra copies
  - True even for memory-mapped I/O
- Software layers add overhead

Storage: Disks, SSDs

Applications
- mmap
- file IO

VFS

Traditional FS

Page cache

Block device

©Copyright 2016 Hewlett Packard Enterprise Company
Non-volatile memory aware file systems

Examples
- Direct access (DAX)
- pmem.io/NVML

Low overhead access to persistent memory
- No page cache
- Direct access with mmap

Linux for The Machine

- HPE's modifications to Linux to support
  - Fabric-attached persistent memory
  - Block device abstractions backed by persistent memory
  - Kernel modifications for “flush on failure”
- Additional support for
  - Fabric-attached memory atomics primitives to handle sharing across SoCs
  - “Librarian File System” for naming and giving permissions to persistent memory
  - Remote virtual memory access
  - Configuration and management utilities

<table>
<thead>
<tr>
<th>Your Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoCE/RDMA Software</td>
</tr>
<tr>
<td>Persistent Memory Support Library</td>
</tr>
<tr>
<td>Fabric-Attached Memory Atomics Library</td>
</tr>
<tr>
<td>Remote Virtual Memory Access (RVMA)</td>
</tr>
<tr>
<td>NVM Backed Block Devices</td>
</tr>
<tr>
<td>Librarian File System</td>
</tr>
<tr>
<td>Librarian File System Controls</td>
</tr>
<tr>
<td>Kernel + Remainder of User Space (L4TM Core)</td>
</tr>
</tbody>
</table>

We plan to contribute all changes back to the Linux community.
SpaceJMP: Programming with Multiple Virtual Address Spaces

- Process has multiple virtual address spaces
  
  **New Process Abstraction:** (PC, registers, VAS*, (VAS))

- Efficient safe programming and sharing for huge memories
- Data sharing and communication between processes
- Versioning and checkpointing
- Co-design between OS, programming languages, compilers, and runtimes
- Prototype implementations in BSD, Linux, and Barrelfish

Memory-driven data management
Traditional databases

– Example: A database (write) transaction

• Traditional databases struggle with big & fast data
• 90% of a database transaction is overhead
• Memory-semantics non-volatile memory: up to 10x improvement

FOEDUS: Fast optimistic engine for data unification services

– Open-source, from-scratch database engine designed to
  – Manipulate data both in DRAM and NVM
  – Take advantage of large multi-core machines
– Fully ACID, serializable database kernel in C++
  – Can be embedded in applications as a library
  – Simplified in-memory applications
– Designed to eliminate scalability bottlenecks
  – Lightweight optimistic concurrency control
  – Decentralized logs are SoC-friendly
  – Design maximizes NVM bandwidth and endurance
FOEDUS: Fast optimistic engine for data unification services
FOEDUS: Open source embedded database

- Scalable up to tens of SoCs
  - Tested scale: Superdome X: 12 TB DRAM, 240 cores
- Efficiently handles datasets larger than DRAM
- Orders of magnitude faster when compared to state-of-the-art in-memory engines

- Open source code, documentation and papers at http://github.com/hkimura/foedus
Memory driven data analytics: Spark for The Machine
Spark contributions
Maximize advantages of large in-memory processing

HPE Components

Spark Core

Artificial Neural Networks
Belief Propagation

Streaming
Machine Learning
SQL
GraphX

API
Off-Heap Memory Store

In-Memory Shuffle Engine

©Copyright 2016 Hewlett Packard Enterprise Company,
In-memory shuffle engine
Provides efficient access to The Machine’s shared-memory architecture and NVM pool

Our approach

– Non-volatile memory based
– In-memory sort/merge
– Optimized CPU cache access

Performance Evaluation for RDD Operators

![Graph showing performance evaluation for RDD Operators]

©Copyright 2016 Hewlett Packard Enterprise Company,
Predictive Analytics
Evaluating Spark for The Machine for fast accurate prediction

Our approach

– Exploit large NVM pool for data caching
– Leverage computationally intensive belief propagation
– Decrease communication cost

15x on HPE Superdome X
SAN FRANCISCO, Calif., — March 1, 2016 — Hortonworks, Inc.® (NASDAQ: HDP) and Hewlett Packard Labs, the central research organization of Hewlett Packard Enterprise® (NYSE: HPE), today announced a new collaboration to enhance Apache Spark, one of the most active Apache big data projects. The collaboration will center around an entirely new class of analytic workloads that benefit from large pools of shared memory.

Early results of the collaboration include the following:

- Enhanced shuffle engine technologies: Faster sorting and in-memory computations, which has the potential to dramatically improve Spark performance.

- Better memory utilization: Improved performance and usage for broader scalability, which will help enable new large-scale use cases.

For more information:

Michelle Lazzar
408-828-9681, comms@hortonworks.com
Memory driven programming models
Do we need separate data representations?

**In-storage durability**

+ Separate object and persistent formats
  – Programmability and performance issues
  – Translation code error-prone and insecure

**In-memory durability**

+ In-memory objects are durable throughout
+ Byte-addressability simplifies programmability
+ Low ld/st latencies offer high performance
  – Persistent does not mean consistent!
NVM-aware application programming
Why can’t I just write my program, and have all my data be persistent?

Consider a simple banking program (just two accounts):

```
double accounts[2];
```

I want to transfer money between accounts. Naïve implementation:

```
transfer(int from, int to, double amount) {
    accounts[from] -= amount;
    accounts[to] += amount;
}
```

Crashes cause corruption, which prevents us from merely restarting the computation
Persistent memory programming – Hewlett Packard Labs

- Manage consistent updates with failure atomicity
- Handle recovery
- Support multi-threaded concurrent access

- Atlas: Persistence for lock-based, multhreaded shared memory programs
  - C/C++11
  - Arbitrary data structures
  - Operate directly on persistent memory within critical sections (lock-based or TM-style transactions)

- Managed Data Structures (MDS) in persistent memory
  - C++11/Java8
  - Specific data structures
  - Library mediated access with ACID transactions with configurable isolation
Manual solution

• Need code that plays back undo log on restart
• Getting this to work with threads and locks is very hard
• Really want to optimize it
• Very unlikely application programmers will get it right
Atlas solution: Consistent sections
Provide a construct that atomically updates NVM

persistent double accounts[2];
transfer(int from, int to, double amount) {
    __atomic {
        accounts[from] -= amount;
        accounts[to]  += amount;
    }
}

• Updates in __atomic block are either completely visible after crash or not at all
• If updates in __atomic block are visible, then so are prior updates to persistent memory

The Atlas programming model
Ensure data consistency in persistent memory

- Programmer distinguishes persistent and transient data
  - Persistent data lives in a “persistent region”
    - E.g., in pseudo-file-system in NVM
    - Directly mapped into process address space (no DRAM buffers)
    - Accessed via CPU loads and stores
  - Programmer writes ordinary multithreaded code
    - Automatic durability support at a fine granularity, complete with recovery code
    - Supports consistency of durable data derived from concurrency constructs

- Open source code available at https://github.com/HewlettPackard/Atlas

Managed Data Structures (MDS)
Simplify programming on persistent in-memory data

– Ease of Programming
  – Programmer manages only application-level data structures
    – MDS data structures are automatically persisted in NVM
  – APIs in multiple programming languages: Java, C++
    – Programmer access through references to data
    – Direct reads and writes
– Ease of Data Sharing
  – Just pass a reference
    – Each program treats the data as if it was local to the program
  – High-level concurrency controls
    – Ensure consistent data in the face of data sharing by multiple threads/processes

Managed Space

Supported data structures: List, Map, Set, Graph, Vector, Queue, . . .

Process 1
Process 2
Java, C++ simultaneously
Supporting safe data sharing

Diagram showing the relationship between parent view, live child view, consumer update transactions, snapshot child view, and business intelligence analytics, with supporting techniques such as non-blocking transactions, zero-copy snapshots, and conflict resolution.
Simplified data management – Benefits to application developer

Conventional Data Formats
- Data structures
- Data format conversion
- Serialisation/deserialisation
- RPC, HTTP, message passing
- Disk communication latency
- Server
- Database
- Filesystem
- Disk

MDS Data Formats
- Data structures
- Local function calls
- Shared non-volatile memory

- Shorter path to persistence
- Less code
- Fewer errors
- Faster development

©Copyright 2016 Hewlett Packard Enterprise Company,
Prototypes and emulators
Show and tell at HPE Discover 2016
Hardware/software co-development

Hardware development

The Simulated/Emulated Machine

The Machine

Software development
Fabric-Attached Memory Emulation for The Machine
Code for memory driven architecture

– Provides a programmer’s view of fabric attached memory
  – QEMU virtual machines running Debian mimic compute nodes
  – Shared memory on the host emulates fabric attached memory (FAM)
  – IVSHMEM links guests and host
– Performant environment allows developers to
  – Create code for The Machine architecture
  – Modify legacy code to take advantage of The Machine architecture
– Open source code available at https://github.com/FabricAttachedMemory
Quartz: NVM Performance Emulator

– Quartz: *performance emulator* for NVM based on commodity hardware (DRAM)
– Focus: modeling primary *performance characteristics of NVM* that affect application end-to-end performance
  – Two performance knobs for NVM emulation: *bandwidth* and *latency*
  – *Non-goals*: accurate simulation of NVM features, NVM functionality, and NVM devices…
– Quartz aims to support:
  – *Sensitivity analysis* of complex applications on future hardware
    – Which ranges of latencies and bandwidth are critical for achieving good performance and scalability?
  – Design questions for future machines with both DRAM and NVM
    – DRAM as cache for NVM vs. DRAM and NVM as peers
    – Strategies for efficient data placement

Open source code available at
https://github.com/HewlettPackard/Quartz


©Copyright 2016 Hewlett Packard Enterprise Company,
Commercial Memory-Driven Computing
HPE Integrity Superdome X

Large-scale shared memory multiprocessor
- Up to 16 processors, 384 cores
  - Intel® Xeon® Processor E7 v4 and E7 v3 family
- Up to 24 TB DDR4 memory (Gen9)
  - 384 DIMM slots
- 24 Mezz PCIe gen3 slots (3 per blade) for IO connectivity to LAN, SAN, and InfiniBand
- 18U enclosure
HPE Persistent Memory

- HPE 8GB NVDIMM Module (782692-B21)
- HPE ProLiant Gen9 Servers Supported and Configurations
  - DL360 Gen9 and DL380 Gen9 E5-2600v4
- Ideal for accelerating databases and analytics workloads
Gen-Z Consortium

- Industry Leaders developing a memory-semantic interconnect

Logos of various companies including AMD, ARM, Broadcom, Cavium, Cray, Dell EMC, Hewlett Packard Enterprise, Huawei, IBM, IDT, Lenovo, Mellanox Technologies, Micron, Microsemi, Red Hat, Samsung, Seagate, SK hynix, WD, and Xilinx.
Communication at the speed of Memory

What is a Memory Semantic Fabric?
- Handles all communication as memory operations such as load/store, put/get and atomic operations typically used by a processor
- Memory semantics are optimal at sub-microsecond latencies from CPU load command to register store
  - Unlike, storage accesses which are block based and managed by complex, code intensive, software stacks

Why Now?
- The emergence of low latency, Storage Class Memory (SCM) and the demand for large capacity, rack scale resource pools, and multi node architectures
Gen-Z: A New Data Access Technology

High Bandwidth
- Memory Semantics – simple Reads and Writes
- From tens to several hundred GB/s of bandwidth
- Sub-100 ns load-to-use memory latency

Low Latency

Advanced Workloads & Technologies
- Real time analytics
- Enables data centric and hybrid computing
- Scalable memory pools for in memory applications
- Abstracts media interface from SoC to unlock new media innovation

Secure Compatible Economical
- Provides end-to-end secure connectivity from node level to rack scale
- Supports unmodified OS for SW compatibility
- Graduated implementation from simple, low cost to highly capable and robust
- Leverages high-volume IEEE physical layers and broad, deep industry ecosystem

Gen-Z Direct Attach, Switched, or Fabric Topology

Compute
- SoC Memory
- SoC Memory

Accelerators
- FPGA Memory
- FPGA Memory
- GPU Memory
- GPU Memory

Pooled Memory

Rack Scale

Open Standard


For open source code...


– Fast optimistic engine for data unification services (FOEDUS): https://github.com/hkimura/foedus


– Fabric Attached Memory Emulation: https://github.com/FabricAttachedMemory/Emulation

– Performance emulation for non-volatile memory latency and bandwidth (Quartz): https://github.com/HewlettPackard/Quartz
Wrapping up

Memory-Driven Computing
• Fast load/store access to large shared pool of fabric-attached non-volatile memory

Many opportunities for software innovation
• Operating systems
• Data stores
• Analytics platforms
• Programming models and tools
• Applications
• Algorithms

How would you exploit Memory-Driven Computing?