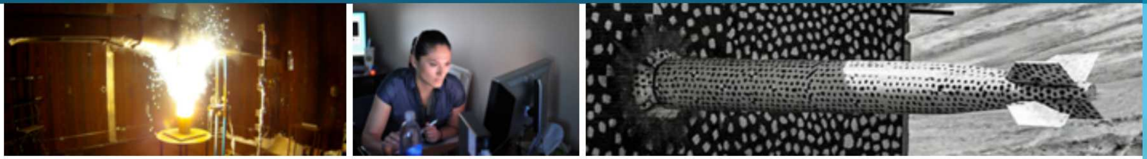
SAND2019-11169C

Opportunities and Challenges for Accelerated Network Interfaces in HPC



Department 1423 - Scalable System Software

Ron Brightwell, R&D Manager

Outline

- Past
  - Brief history of network offload in HPC
- Present
  - SmartNICs
- Future
  - Recent work on network offload technologies
  - Disaggregated system architecture

A Brief History of Network Offload in HPC

Center for Computing Research

Intel Paragon Node (Circa 1993)

Intel Paragon™ Supercomputers

Paragon™ Supercomputers set a new standard for performance, programmability, and practicality in high-performance computing.

Designed to solve today's most challenging computational problems, Intel Paragon™ Supercomputers set new standards for sustained performance and usability in scalable, high-performance computing systems. Paragon XP/S supercomputers provide the overall capacity, balanced performance and ease-of-use needed to solve Grand Challenge computing problems, while offering the system services and administrative tools to manage the demanding environment of even the largest supercomputer centers. Paragon XP/E systems are entry-level supercomputers aimed at small institutions needing low-cost scalable systems for teaching and R&D, as well as consortium members needing local development platforms for larger, remote Paragon machines.

PARAGON SUPERCOMPUTERS

High Performance
- Configurations of 0.6 to 36 GFLOPS, 0.35 to 21 KMIPS
- Hardware node-to-node communications at up to 175 MB/sec, full duplex per channel
- Aggregate hardware interconnect bisection bandwidth to 5.6 GB/sec
- Scalable main memory to 77 GB†
- Scalable internal disk storage to 272 GB

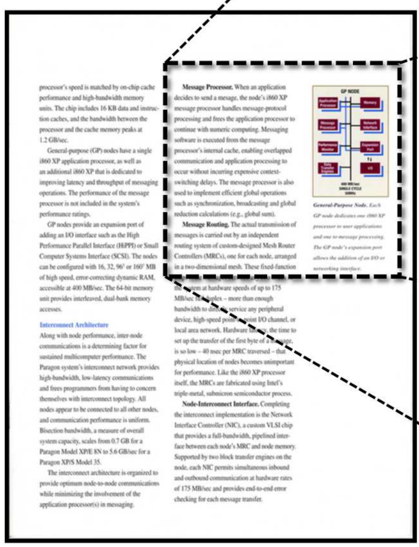
Production Ready
- Industry-standard OSF®/1 operating system
- Multiple HiPPI, Ethernet, FDDI connections
- Simultaneous batch and interactive operation
- Resource control, accounting and scheduling tools
- Designed for high availability
- Air-cooled operation

Enhanced Programmability
- Fortran-77, C, Ada
- High Performance Fortran††, C++††
- ParAide interactive, graphical development environment
- Performance-optimized node libraries and parallel equation solvers
- X Window System®, DGL®, OpenGL® and OSF/Motif® graphics
- Compatibility with Intel iPSC®/860 family

System Overview

The Paragon supercomputer is a distributed-memory multicomputer based on Intel's teraFLOPS architecture, which has been developed over the course of four generations of parallel supercomputers and is implemented using Intel's advanced microprocessor and

intel



processor's speed is matched by on-chip cache performance and high-bandwidth memory units. The chip includes 16 KB data and instruction caches, and the bandwidth between the processor and the cache memory peaks at 1.2 GB/sec.

General-purpose (GP) nodes have a single i860 XP application processor, as well as an additional i860 XP that is dedicated to improving latency and throughput of messaging operations. The performance of the message processor is not included in the system's performance ratings.

GP nodes provide an expansion port of adding an I/O interface such as the High Performance Parallel Interface (HiPPI) or Small Computer Systems Interface (SCSI). The nodes can be configured with 16, 32, 96† or 160† MB of high speed, error-correcting dynamic RAM, accessible at 400 MB/sec. The 64-bit memory unit provides interleaved, dual-bank memory accesses.

Interconnect Architecture

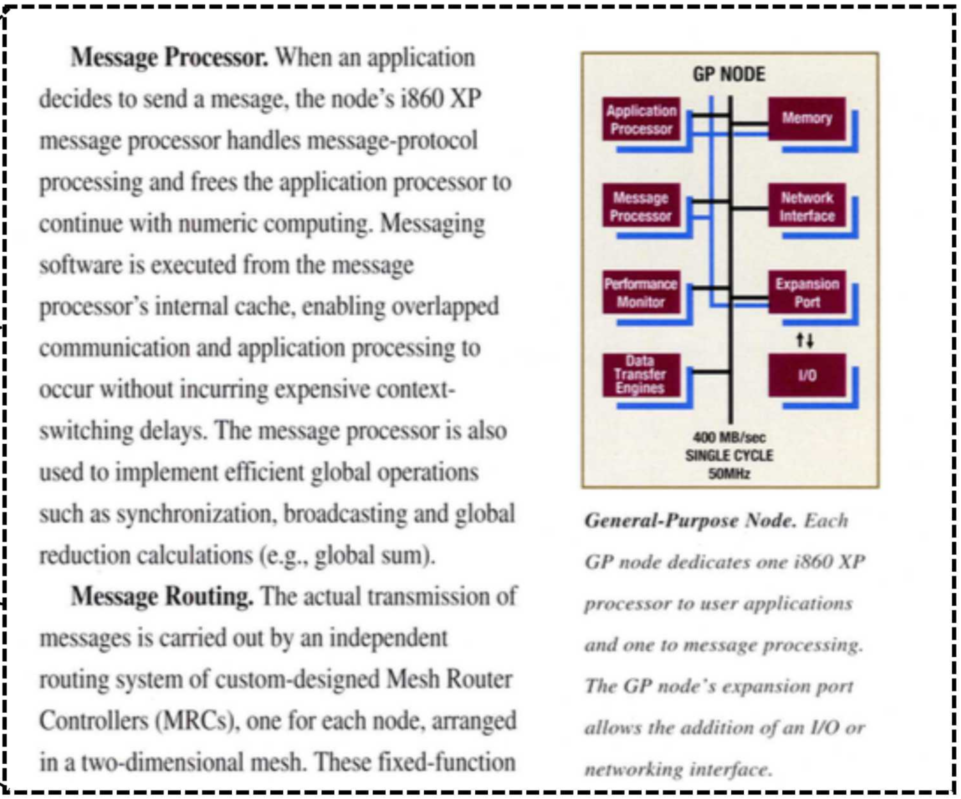
Along with node performance, inter-node communications is a determining factor for sustained multicomputer performance. The Paragon system's interconnect network provides high-bandwidth, low-latency communications and frees programmers from having to concern themselves with interconnect topology. All nodes appear to be connected to all other nodes, and communication performance is uniform. Bisection bandwidth, a measure of overall system capacity, scales from 0.7 GB for a Paragon Model XP/E 8N to 5.6 GB/sec for a Paragon XP/S Model 35.

The interconnect architecture is organized to provide optimum node-to-node communications while minimizing the involvement of the application processor(s) in messaging.

Message Processor. When an application decides to send a mesage, the node's i860 XP message processor handles message-protocol processing and frees the application processor to continue with numeric computing. Messaging software is executed from the message processor's internal cache, enabling overlapped communication and application processing to occur without incurring expensive context-switching delays. The message processor is also used to implement efficient global operations such as synchronization, broadcasting and global reduction calculations (e.g., global sum).

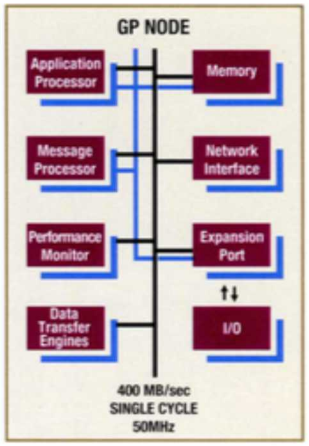
Message Routing. The actual transmission of messages is carried out by an independent routing system of custom-designed Mesh Router Controllers (MRCs), one for each node, arranged in a two-dimensional mesh. These fixed-function [illegible] the system at hardware speeds of up to 175 MB/sec full-duplex – more than enough bandwidth to directly service any peripheral device, high-speed point-to-point I/O channel, or local area network. Hardware latency, the time to set up the transfer of the first byte of a message, is so low – 40 nsec per MRC traversed – that physical location of nodes becomes unimportant for performance. Like the i860 XP processor itself, the MRCs are fabricated using Intel's triple-metal, submicron semiconductor process.

Node-Interconnect Interface. Completing the interconnect implementation is the Network Interface Controller (NIC), a custom VLSI chip that provides a full-bandwidth, pipelined interface between each node's MRC and node memory. Supported by two block transfer engines on the node, each NIC permits simultaneous inbound and outbound communication at hardware rates of 175 MB/sec and provides end-to-end error checking for each message transfer.



Message Processor. When an application decides to send a mesage, the node's i860 XP message processor handles message-protocol processing and frees the application processor to continue with numeric computing. Messaging software is executed from the message processor's internal cache, enabling overlapped communication and application processing to occur without incurring expensive context-switching delays. The message processor is also used to implement efficient global operations such as synchronization, broadcasting and global reduction calculations (e.g., global sum).

Message Routing. The actual transmission of messages is carried out by an independent routing system of custom-designed Mesh Router Controllers (MRCs), one for each node, arranged in a two-dimensional mesh. These fixed-function



General-Purpose Node. *Each GP node dedicates one i860 XP processor to user applications and one to message processing. The GP node's expansion port allows the addition of an I/O or networking interface.*

Intel ASCI Red (TFLOPS) Compute Node

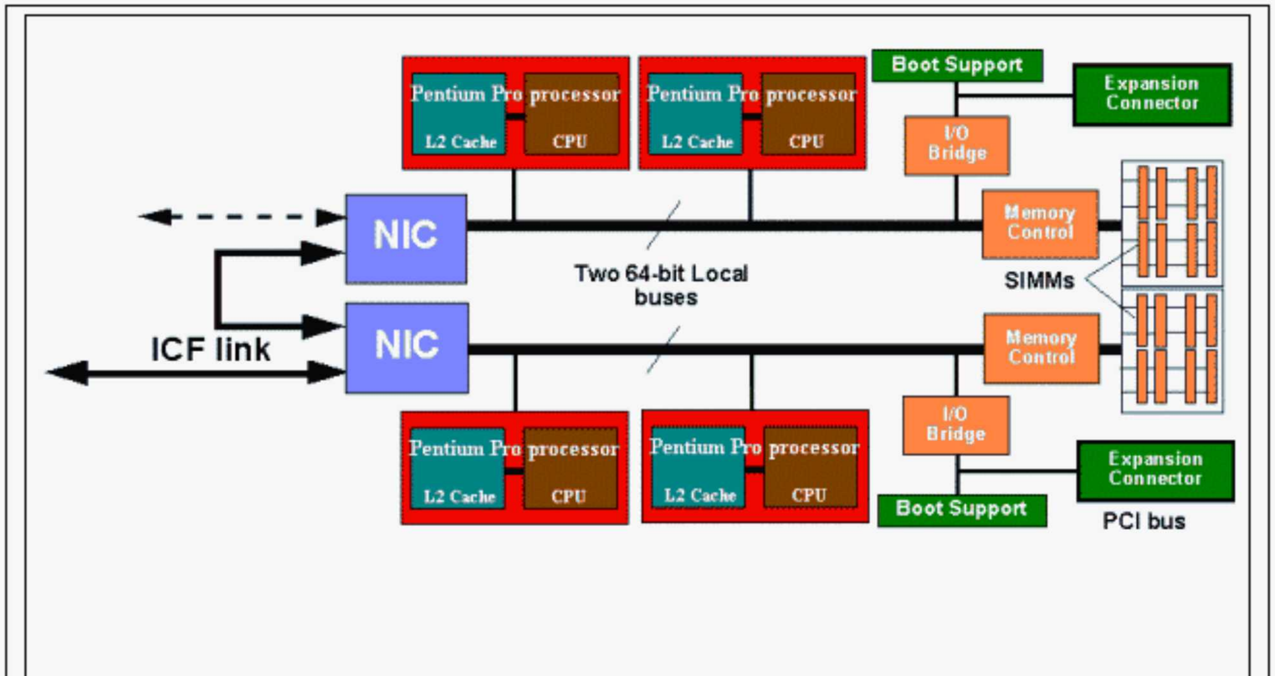


Figure 3: The ASCI Option Red supercomputer Kestrel Board. This board includes two compute nodes chained together through their NIC's. One of the NIC's connects to the MRC on the backplane through the ICF Link.

Paragon/TFLOPS Network Interface Controller (NIC)

- Attached to the memory bus
- Cache coherent with the processor(s)
- Programmed by the operating system (OS)
  - Device driver was embedded in the OS
  - Driver consisted of programming DMA engines and memory-mapped registers
- Interrupt-driven
  - An interrupt would be generated for:
    - Arrival of an incoming message
    - Completion of an incoming message
    - Completion of an outgoing message
- Messages initiation via system call trap
- Source-routed, circuit-switched, wormhole routed network
  - Message header contained route to destination
  - Message body was one contiguous block

Basic Assumptions About Networking for Massively Parallel Systems

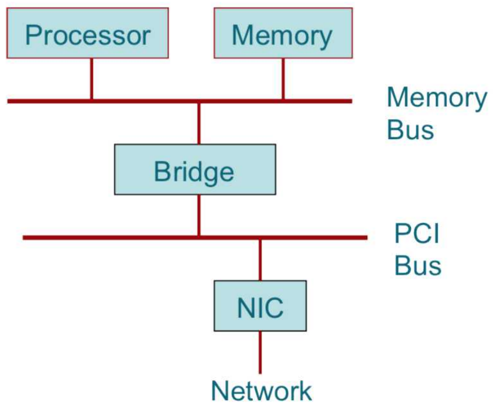
- A single low-level network API is needed
  - Compute node may not have a TCP/IP stack
  - System is space-shared
    - Compute node application should own all network resources
- Applications will use multiple protocols simultaneously
  - Can't focus on just MPI
  - Runtime system, system call forwarding, I/O protocols too
- Need to support communication between unrelated processes
  - Client/server communication between application processes and system services
- Need to support general-purpose interconnect capabilities
  - Can't assume special collective network hardware
- Interconnect hardware limitations can't be fixed in software

Key Network Capabilities

- Independent progress
  - Data should move without requiring polling from user-level library
  - Adhere to the strong progress rule interpretation of MPI
- Overlap
  - Decouple the host processors from the network as much as possible
  - Enable overlap of computation and communication as well as communication and communication
- Scalable use of memory resources
  - Buffer space for MPI unexpected messages
  - Memory use should be independent of the number of peers
- High performance
  - Maximize bandwidth by avoiding memory-to-memory copies
  - Minimize latency by avoiding OS interaction

Programmable User-Level Networks Enabled API Exploration

- Myrinet (~1994)
  - First commercially available Gb/s standalone network
  - Based on technology developed for Intel MPP networks
  - Initially available for Sun SPARC SBus, later for PCI-based PCs
  - Custom embedded MIPS-based programmable processor (LANai)
  - Myrinet Control Program (MCP) software development environment
  - Destination routed, maximum message size (packets)
  - Numerous APIs and MCPs: AM, FM, GM, PM, MX
- Quadrics QSNet (Elan + Elite) (~2001)
  - Outgrowth of technology developed for Meiko MPP networks
  - Offered several different APIs for user-level networking
  - Provided a development environment for running user-level functions on NIC



Portals Interconnect Programming Interface
A Vehicle for Hardware/Software Co-Design

- Initially developed primarily by Sandia and the University of New Mexico
- Deployed on several production massively parallel processing (MPP) and cluste
  - 1993: 1800-node Intel Paragon (SUNMOS)
  - 1997: 10,000-node Intel ASCI Red (Puma/Cougar)
  - 1999: 1800-node Cplant cluster (Linux)
  - 2005: 10,000-node Cray Sandia Red Storm (Catamount)
  - 2009: 18,688-node Cray XT5 – ORNL Jaguar (Linux)
  - 2017: Bull BXI interconnect
- Focused on providing
  - Lightweight “connectionless” model for massively parallel systems
  - Low latency, high bandwidth
  - Independent progress
  - Overlap of computation and communication
  - Scalable buffering semantics
  - Protocol building blocks to support higher-level application protocols and libraries and system services

http://www.cs.sandia.gov/Portals/

Fixing Semantic Mismatch Between Layers

Majority of interconnect software R&D is spent on dealing with the semantic mismatch between what the upper-layer protocols need and what the low-level network software and the underlying hardware provide

RDMA (e.g. InfiniBand Verbs)

One-sided
- Allows process to read/write remote memory implicitly

Zero-copy data transfer
- No need for intermediate buffering in host memory

Low CPU overhead
- Decouples host processor from network

Fixed memory resources
- No unexpected Messages

Supports unstructured, non-blocking data transfer
- Completion is a local event

MPI Point-to-Point

Two-sided
- Short messages are copied
- Long messages need rendezvous

CPU involved in every message
- Message matching

Unexpected messages
- Need flow control

Completion may be non-local
- Need control messages

Motivation for Low-Level Transport APIs

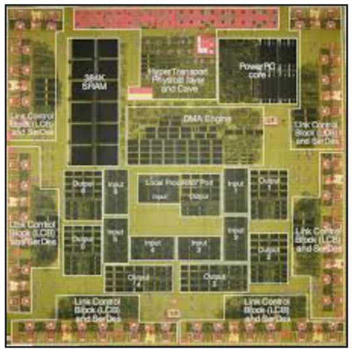
- Targeting a single programming model target is too limiting (top down approach)
  - MPI - MPICH, OpenMPI
  - PGAS - GasNet, OpenSHMEM
  - I/O – LNet, Mercury, Nessie
- Desire to reduce development costs
  - Provide one network abstraction for all ULPs
  - Large porting effort is a strong indication of the semantic mismatch
- “Vendor differentiation
  - Which really defeats the portability goal
- Vehicle for hardware/software co-design R&D

Red Storm – Prototype for Cray XT Series

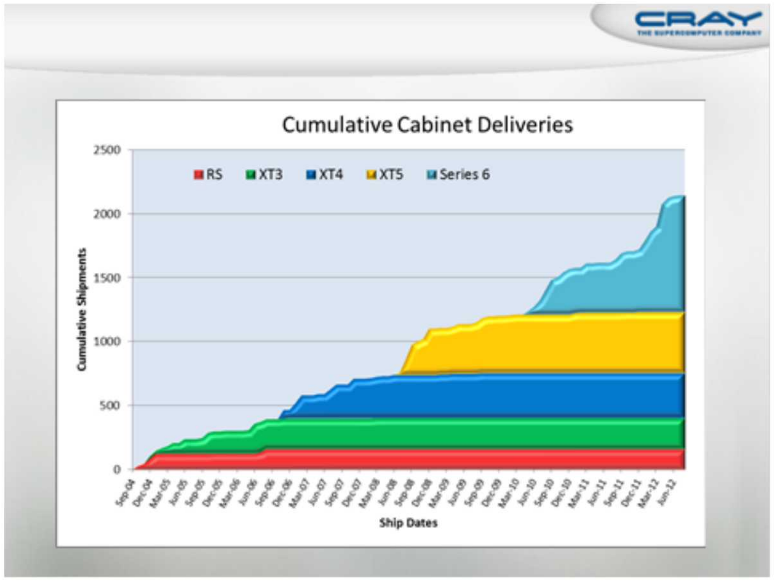
- Architected by Sandia, engineered jointly with Cray in 2003
  - Sandia contributed to the design of the SeaStar network interface and router

Sandia also developed
  - Lightweight kernel compute node OS
  - Scalable parallel job launching system
  - Portals high-performance interconnect programming interface
  - SeaStar firmware

- 140+ systems to 80 different customers worldwide
  - Including ORNL, NERSC, and LANL
- Following Red Storm, Cray's market share rose from 6% in 2002 to 21% in 2007*
- Revenue of $1B +
- Basis of Cray's business today
- "Virtually everything we do at Cray - each of our three business units - comes from Red Storm. It spawned a company around it, a historic company struggling as to where we would go next. Literally, this program saved Cray." – Pete Ungaro, CEO, Cray Inc.
- https://www.datacenterdynamics.com/analysis/after-the-storm-the-supercomputer-that-saved-cray/


384K SRAM
HyperTransport Physical layer and Cave
PowerPC core
DMA Engine
Link Control Block (LCB) and SerDes

SeaStar was a PowerPC-Based System-on-a-Chip (SOC)



*Source: IDC #209251 *Technical Computing Systems: Competitive Analysis*, November 2007

Significant Vendor Impact of Sandia's Portals Networking Technology

All of these production vendor-supported systems used Portals as the network hardware programming interface.
Portals enabled the first TeraFLOPS platform (ASCI Red) and the first non-accelerated PetaFLOPS platform (Jaguar).

Intel Paragon

Portals 0

Intel ASCI Red

Portals 2

Cray Red Storm

Portals 3

Cray XT3, XT4, XT5

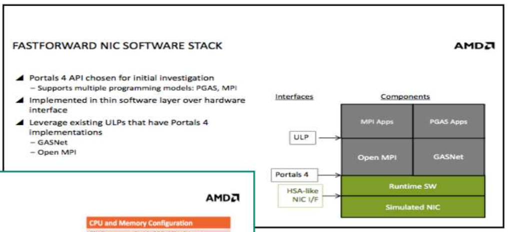
Portals 3

Atos Tera1000

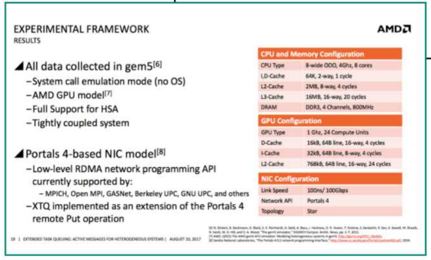
Portals 4

Unlike other low-level network programming interfaces, Portals is intended to enable co-design rather than serve as a portability layer.
The influence and impact of Portals can be seen in vendor co-design activities, other low-level network programming interfaces, and emerging network hardware.

AMD FastForward Project based on Portals 4


FASTFORWARD NIC SOFTWARE STACK

- Portals 4 API chosen for initial investigation
  - Supports multiple programming models: PGAS, MPI
- Implemented in thin software layer over hardware interface
- Leverage existing ULPs that have Portals 4 implementations
  - GASNet
  - Open MPI


EXPERIMENTAL FRAMEWORK

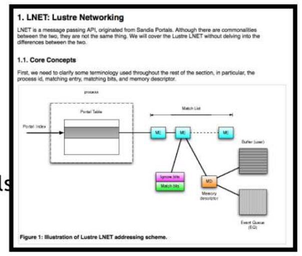
RESULTS

- All data collected in gem5[6]
  - System call emulation mode (no OS)
  - AMD GPU model[7]
  - Full Support for HSA
  - Tightly coupled system
- Portals 4-based NIC model[8]
  - Low-level RDMA network programming API currently supported by:
    - MPICH, Open MPI, GASNet, Berkeley UPC, GNU UPC, and others
  - XTQ implemented as an extension of the Portals 4 remote Put operation

CPU and Memory Configuration	
CPU Type	8-wide OOO, 4Ghz, 8 cores
I,D-Cache	64K, 2-way, 1 cycle
L2-Cache	2MB, 8-way, 4 cycles
L3-Cache	16MB, 16-way, 20 cycles
DRAM	DDR3, 4 Channels, 800MHz
GPU Configuration	
GPU Type	1 Ghz, 24 Compute Units
D-Cache	16kB, 64B line, 16-way, 4 cycles
I-Cache	32kB, 64B line, 8-way, 4 cycles
L2-Cache	768kB, 64B line, 16-way, 24 cycles
NIC Configuration	
Link Speed	100ns/ 100Gbps
Network API	Portals 4
Topology	Star

19 | EXTENDED TASK QUEUING: ACTIVE MESSAGES FOR HETEROGENEOUS SYSTEMS | AUGUST 10, 2017

Lustre File System network based on Portals 4


1. LNET: Lustre Networking

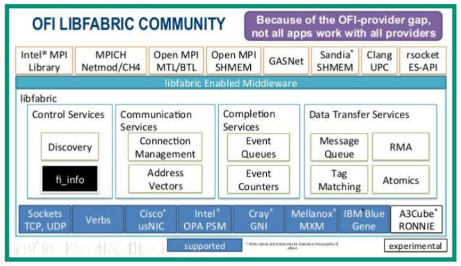
LNET is a message passing API, originated from Sandia Portals. Although there are commonalities between the two, they are not the same thing. We will cover the Lustre LNET without delving into the differences between the two.

1.1. Core Concepts

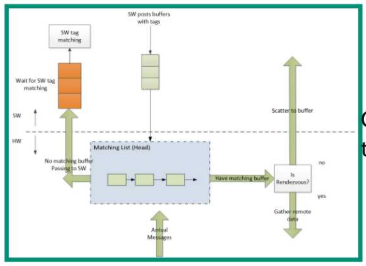
First, we need to clarify some terminology used throughout the rest of the section, in particular, the process id, matching entry, matching bits, and memory descriptor.

Figure 1: Illustration of Lustre LNET addressing scheme.

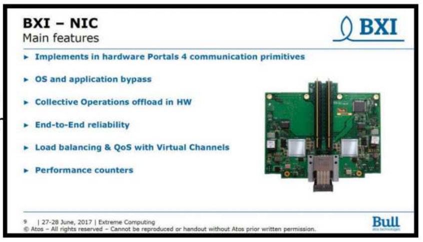
OFI Libfrabric API based on Portals 4



Mellanox ConnectX-5 MPI tag matching in hardware



Atos Bull eXascale Interconnect (BXI) based on Portals 4


BXI – NIC

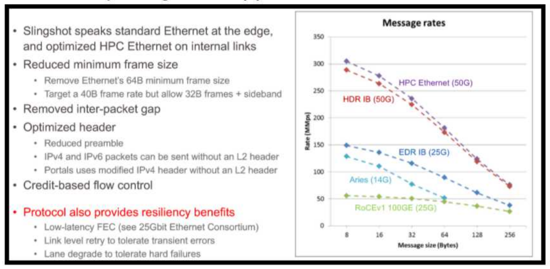
Main features

- Implements in hardware Portals 4 communication primitives
- OS and application bypass
- Collective Operations offload in HW
- End-to-End reliability
- Load balancing & QoS with Virtual Channels
- Performance counters

9 | 27-28 June, 2017 | Extreme Computing

© Atos – All rights reserved – Cannot be reproduced or handout without Atos prior written permission.

Cray Slingshot Supports Portals 4 header

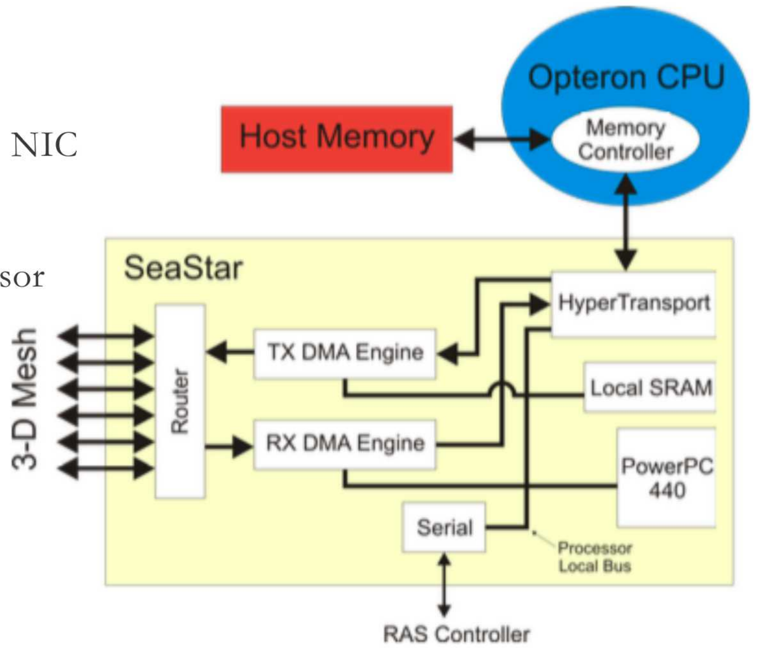

- Slingshot speaks standard Ethernet at the edge, and optimized HPC Ethernet on internal links
- Reduced minimum frame size
  - Remove Ethernet's 64B minimum frame size
  - Target a 40B frame rate but allow 32B frames + sideband
- Removed inter-packet gap
- Optimized header
  - Reduced preamble
  - IPv4 and IPv6 packets can be sent without an L2 header
  - Portals uses modified IPv4 header without an L2 header
- Credit-based flow control
- Protocol also provides resiliency benefits
  - Low-latency FEC (see 25Gbit Ethernet Consortium)
  - Link level retry to tolerate transient errors
  - Lane degrade to tolerate hard failures

SmartNICs

Center for Computing Research

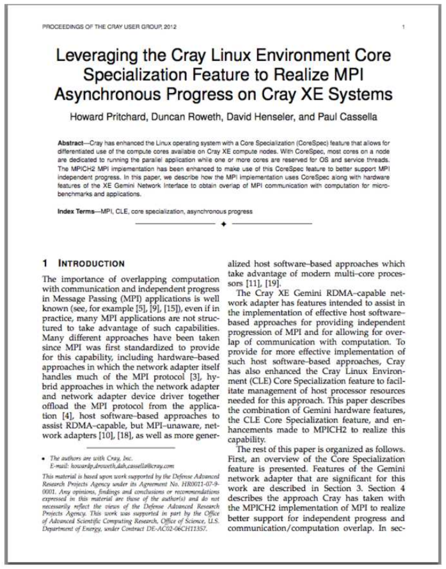
Onload Versus Offload Argument (~2005)

- Why design a custom NIC for offload?
- It is too expensive
  - HPC market is too small to justify a custom ASIC for an HPC NIC
- Just dedicate a core
  - A 3 GHz Xeon will outperform a 500 MHz embedded processor on network protocol processing
  - Overhead cost will go down as core count increases
  - Cores won't be getting slower, right?



Cray Core Specialization

- Cray's chose a complete re-design for Gemini NIC
  - Gemini was delivered six years after the first SeaStar chip
- Dedicate "OS" cores to handle MPI progress
  - MPI progress threads run on a dedicated set of cores



PROCEEDINGS OF THE CRAY USER GROUP, 2012 1

Leveraging the Cray Linux Environment Core Specialization Feature to Realize MPI Asynchronous Progress on Cray XE Systems

Howard Pritchard, Duncan Roweth, David Henseler, and Paul Cassella

Abstract—Cray has enhanced the Linux operating system with a Core Specialization (CoreSpec) feature that allows for differentiated use of the compute cores available on Cray XE compute nodes. With CoreSpec, most cores on a node are dedicated to running the parallel application while one or more cores are reserved for OS and service threads. The MPICH2 MPI implementation has been enhanced to make use of this CoreSpec feature to better support MPI independent progress. In this paper, we describe how the MPI implementation uses CoreSpec along with hardware features of the XE Gemini Network Interface to obtain overlap of MPI communication with computation for micro-benchmarks and applications.

Index Terms—MPI, CLE, core specialization, asynchronous progress

1 INTRODUCTION

The importance of overlapping computation with communication and independent progress in Message Passing (MPI) applications is well known (see, for example [5], [9], [15]), even if in practice, many MPI applications are not structured to take advantage of such capabilities. Many different approaches have been taken since MPI was first standardized to provide for this capability, including hardware–based approaches in which the network adapter itself handles much of the MPI protocol [3], hybrid approaches in which the network adapter and network adapter device driver together offload the MPI protocol from the application [4], host software–based approaches to assist RDMA–capable, but MPI–unaware, network adapters [10], [18], as well as more generalized host software–based approaches which take advantage of modern multi–core processors [11], [19].

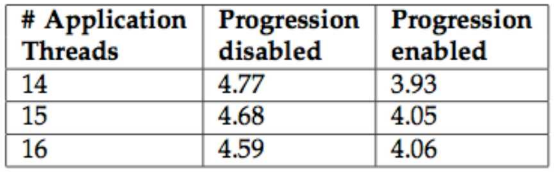
The Cray XE Gemini RDMA–capable network adapter has features intended to assist in the implementation of effective host software–based approaches for providing independent progression of MPI and for allowing for overlap of communication with computation. To provide for more effective implementation of such host software–based approaches, Cray has also enhanced the Cray Linux Environment (CLE) Core Specialization feature to facilitate management of host processor resources needed for this approach. This paper describes the combination of Gemini hardware features, the CLE Core Specialization feature, and enhancements made to MPICH2 to realize this capability.

The rest of this paper is organized as follows. First, an overview of the Core Specialization feature is presented. Features of the Gemini network adapter that are significant for this work are described in Section 3. Section 4 describes the approach Cray has taken with the MPICH2 implementation of MPI to realize better support for independent progress and communication/computation overlap. In sec-

• *The authors are with Cray, Inc.*
E-mail: howardp,droweth,dah,cassella@cray.com

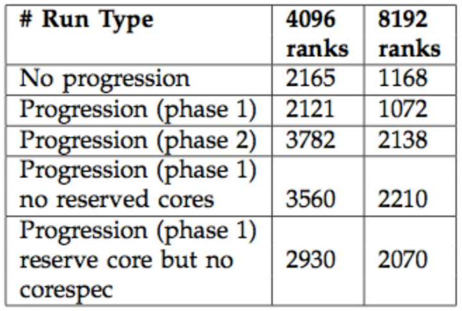
This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency. This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

S3D Time Step Summary



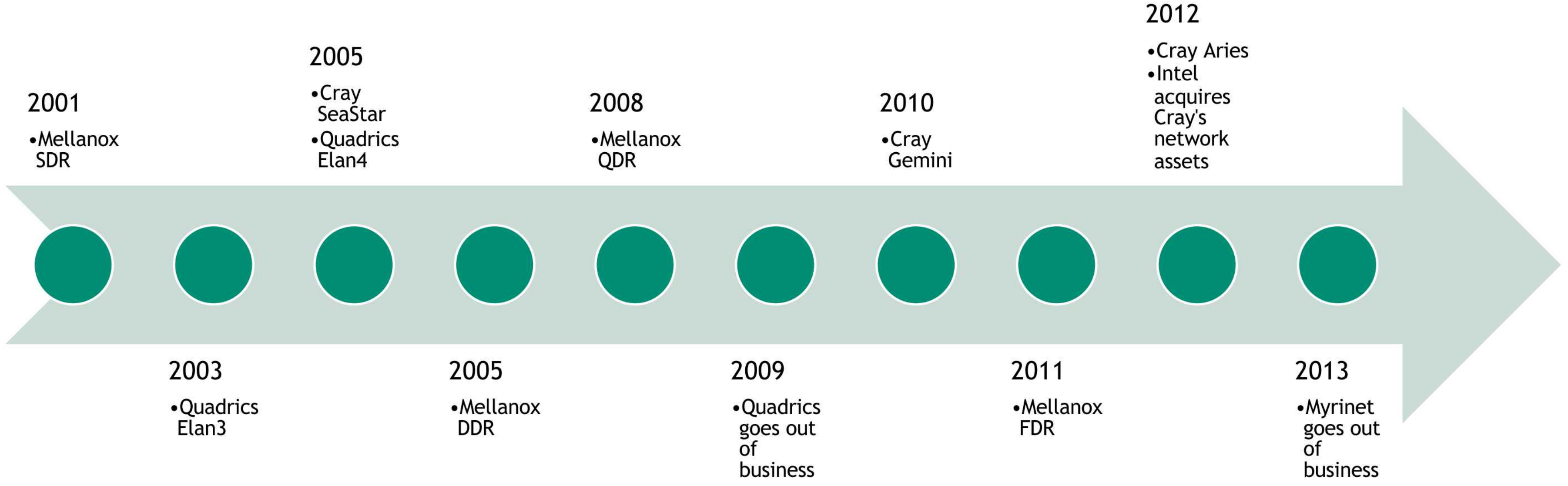
# Application Threads	Progression disabled	Progression enabled
14	4.77	3.93
15	4.68	4.05
16	4.59	4.06

MILC Run Time Summary(secs)



# Run Type	4096 ranks	8192 ranks
No progression	2165	1168
Progression (phase 1)	2121	1072
Progression (phase 2)	3782	2138
Progression (phase 1) no reserved cores	3560	2210
Progression (phase 1) reserve core but no corespec	2930	2070

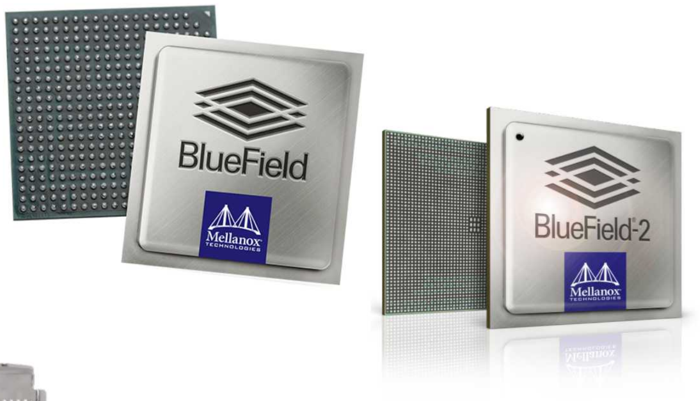
HPC NIC Timeline

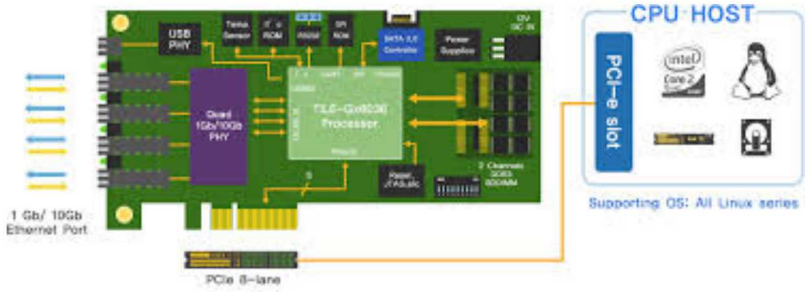


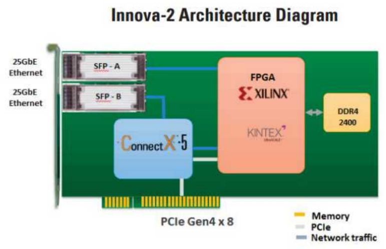
2014 - 2017

- NDAs prohibit me from saying much about this time period
- Two offload capable HPC NIC in design/development
  - Bull BXI
- Offload-capable and programmable NICs aren't fashionable

2019 - SmartNICs are Everywhere





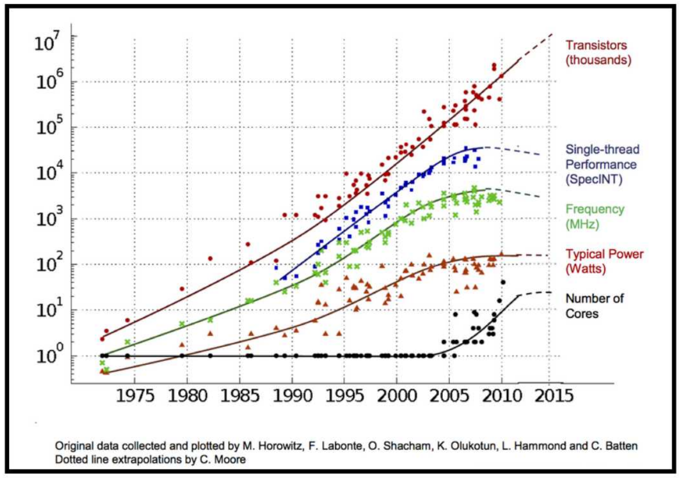


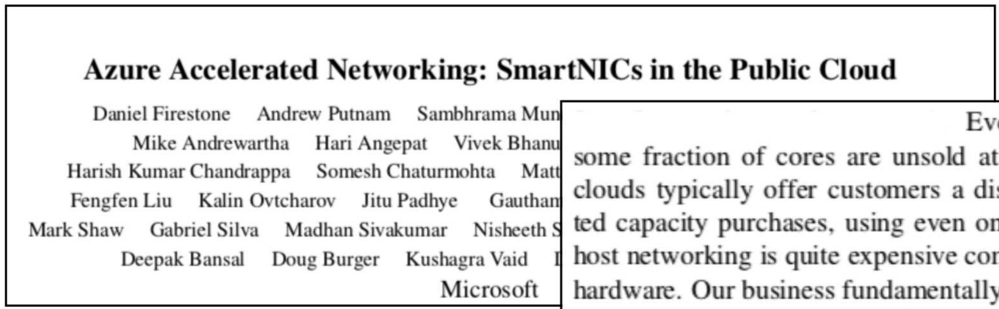
CN2350-210SVPN

CN7890-240NV

What Happened?

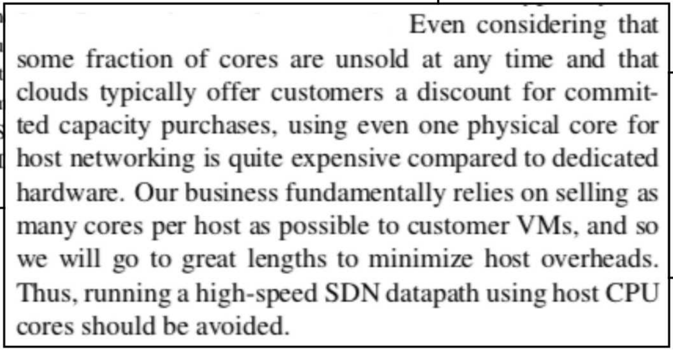
- Core frequency stalled and signaling rate kept going
  - A single core doing network protocol processing can't keep up
- Cloud computing with virtual machines
  - SDN, Open vSwitch, DPDK
- Main barrier to developing offload capable NICs was not technical
  - Economics drive innovation



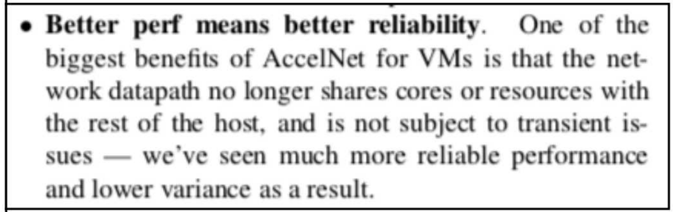


Azure Accelerated Networking: SmartNICs in the Public Cloud

Daniel Firestone Andrew Putnam Sambhrama Mun
Mike Andrewartha Hari Angepat Vivek Bhanu
Harish Kumar Chandrappa Somesh Chaturmohta Matt
Fengfen Liu Kalin Ovtcharov Jitu Padhye Gautham
Mark Shaw Gabriel Silva Madhan Sivakumar Nisheeth S
Deepak Bansal Doug Burger Kushagra Vaid I
Microsoft



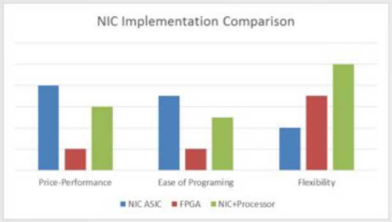
Even considering that some fraction of cores are unsold at any time and that clouds typically offer customers a discount for committed capacity purchases, using even one physical core for host networking is quite expensive compared to dedicated hardware. Our business fundamentally relies on selling as many cores per host as possible to customer VMs, and so we will go to great lengths to minimize host overheads. Thus, running a high-speed SDN datapath using host CPU cores should be avoided.



- **Better perf means better reliability**. One of the biggest benefits of AccelNet for VMs is that the network datapath no longer shares cores or resources with the rest of the host, and is not subject to transient issues — we've seen much more reliable performance and lower variance as a result.

What is a SmartNIC?

- Designed for virtualization environments and SDN
- Need significant more processing for network protocol offload
- Programmable from the network protocol perspective
  - Not really intended to offload upper-level protocol functionality
- Move parallelization into the NIC because they can't get it in the host
  - Flows can be parallelized more easily below the VM
- Tradeoffs between ASIC, FPGA, SOC
  - FPGAs provide a more flexible platform for rapidly evolving TLAs
  - FPGAs provide more reliable performance and are not subject to variance



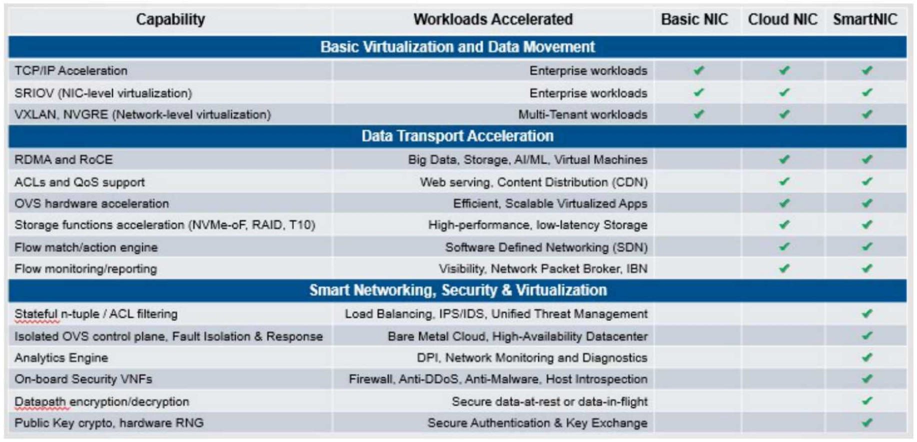
*Implementation does **Not** make a NIC a SmartNIC*

- ASIC Based
  - Excellent price-performance
  - Vendor development cost high
  - Programmable and extensible
    - Easy to program but flexibility is limited to pre-defined capabilities

- FPGA Based
  - Good performance but expensive
  - Very difficult to program
  - Workload specific optimization

- SOC Based

  System on Chip - NIC + CPU
  - Good price-performance
  - C Programmable Processors
  - Highest Flexibility
  - Easiest programmability



Capability	Workloads Accelerated	Basic NIC	Cloud NIC	SmartNIC
Basic Virtualization and Data Movement				
TCP/IP Acceleration	Enterprise workloads	✓	✓	✓
SRIOV (NIC-level virtualization)	Enterprise workloads	✓	✓	✓
VXLAN, NVGRE (Network-level virtualization)	Multi-Tenant workloads	✓	✓	✓
Data Transport Acceleration				
RDMA and RoCE	Big Data, Storage, AI/ML, Virtual Machines		✓	✓
ACLs and QoS support	Web serving, Content Distribution (CDN)		✓	✓
OVS hardware acceleration	Efficient, Scalable Virtualized Apps		✓	✓
Storage functions acceleration (NVMe-oF, RAID, T10)	High-performance, low-latency Storage		✓	✓
Flow match/action engine	Software Defined Networking (SDN)		✓	✓
Flow monitoring/reporting	Visibility, Network Packet Broker, IBN		✓	✓
Smart Networking, Security & Virtualization				
Stateful n-tuple / ACL filtering	Load Balancing, IPS/IDS, Unified Threat Management			✓
Isolated OVS control plane, Fault Isolation & Response	Bare Metal Cloud, High-Availability Datacenter			✓
Analytics Engine	DPI, Network Monitoring and Diagnostics			✓
On-board Security VNFs	Firewall, Anti-DDoS, Anti-Malware, Host Introspection			✓
Datapath encryption/decryption	Secure data-at-rest or data-in-flight			✓
Public Key crypto, hardware RNG	Secure Authentication & Key Exchange			✓

Recent Work on Network Offload Technologies

Center for Computing Research

Active Messages (AM)

- T. von Eicken, et al.: "Active Messages: A Mechanism for Integrated Communication and Computation" (1992)
- Lots of different flavors
  - Pure active messages
    - Origin sends message to target containing code and data
    - Target invokes code on that data
  - Generalized active messages
    - Origin sends message to target containing function id and data
    - Function id maps to existing code in target's address space
- Similar to remote procedure invocation without returning a result to origin
- Semantically equivalent to blocking a thread on an incoming message and invoking a handler when the message arrives

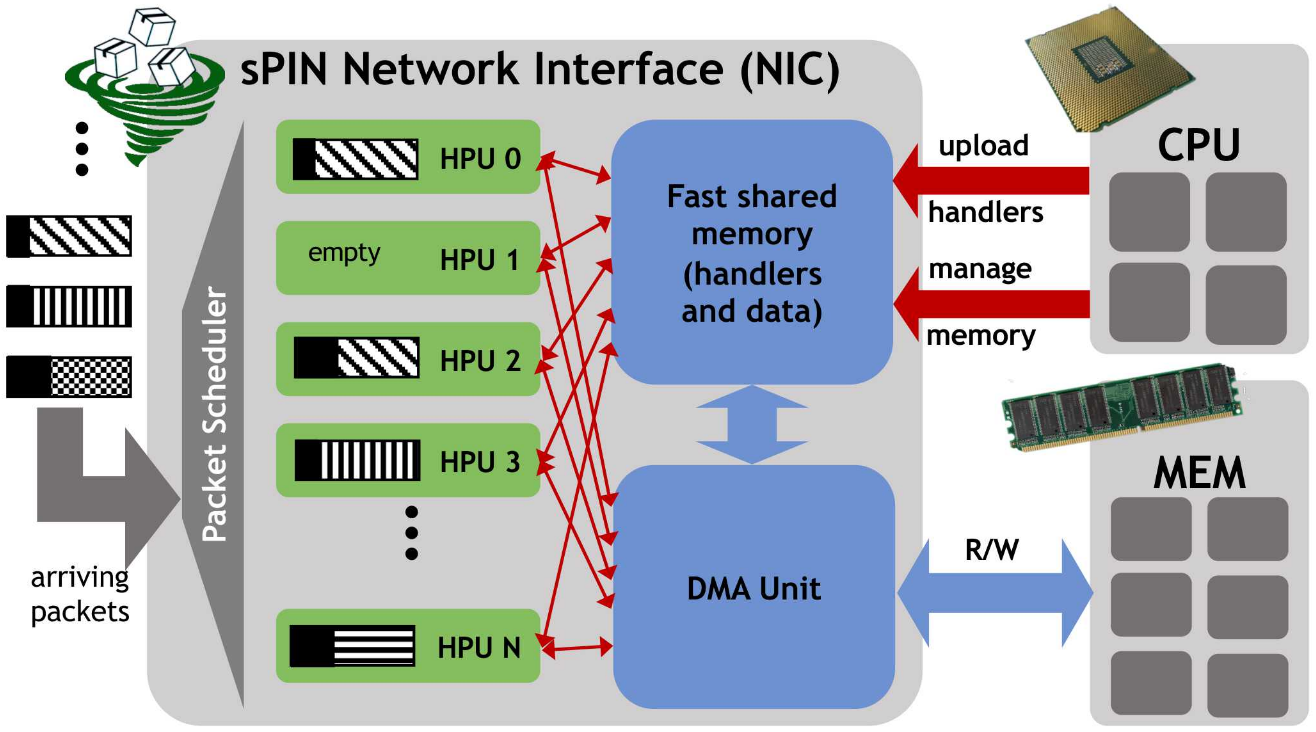
Issues with Active Messages

- Data delivery
  - Who determines where the data goes – origin or target?
  - How much data can be delivered?
- Handlers
  - When are resources allocated – a priori or on arrival?
  - What can be called?
  - Where do they run (context)?
  - When do they run relative to message delivery?
  - How long do they run?
  - Why?
    - One-sided messages decouple processor from network
    - Active messages tightly couple processor and network
      - Active messages aren't one-sided
      - Memory is the endpoint, not the cores
    - Lightweight mechanism for sleeping/waking thread on memory update
      - Why go through the network API for this?
- Scheduling lots of unexpected thread invocations leads to flow control issues

Is There a Better Way to Get AM Semantics?

- Cores are slower, more energy-efficient
  - Modern cores require 15-20 ns to access L3 cache
    - Haswell – 34 cycles
    - Skylake - 44 cycles
- Terabit per second networks are coming
  - 400 Gib/s can deliver a 64-byte message every 1.2 ns
- Need to remove processor from network processing path (offload)
- RDMA only supports data transfer between virtual memory spaces
  - Data is placed blindly into memory
  - Need varying levels of steering the data at the target

streaming Processing In the Network (sPIN)



Hoefler, Di Girolamo, Taranov, Grant, Brightwell. “sPIN: High-Performance Streaming Processing in the Network,” in Proceedings of SC”17, November 2017.

sPIN is not Active Messages

- Tightly integrated NIC packet processing
- AMs are invoked on full messages
  - sPIN works on packets
  - Allows for pipelining packet processing
- AM uses host memory for buffering messages
  - sPIN stores packets in fast buffer memory on the NIC
  - Accesses to host memory are allowed but should be minimized
- AM messages are atomic
  - sPIN packets can be processed atomically

sPIN Approach

- Handlers are executed on NIC Handler Processing Units (HPUs)
- Simple runtime manages HPUs
- Each handler owns shared memory that is persistent for the lifetime of a message
  - Handlers can use this memory to keep state and communicate
- NIC identifies all packets belonging to the same message
- Three handler types
  - Header handler – first packet in a message
  - Payload handler – all subsequent packets
  - Completion handler – after all payload handlers complete
- HPU memory is managed by the host OS
- Host compiles and offloads handler code to the HPU
- Handler code is only a few hundred instructions

sPIN Approach (cont'd)

- Handlers are written in standard C/C++ code
- No system calls or complex libraries
- Handlers are compiled to the specific Network ISA
- Handler resources are accounted for on a per-application basis
  - Handlers that run too long may stall NIC or drop packets
- Programmers need to ensure handlers run at line rate
- Handlers can start executing within a cycle after packet arrival
  - Assuming an HPU is available
- Handlers execute in a sandbox relative to host memory
  - They can only access application's virtual address space
  - Access to host memory is via DMA

INCA: In-Network Compute Assistance

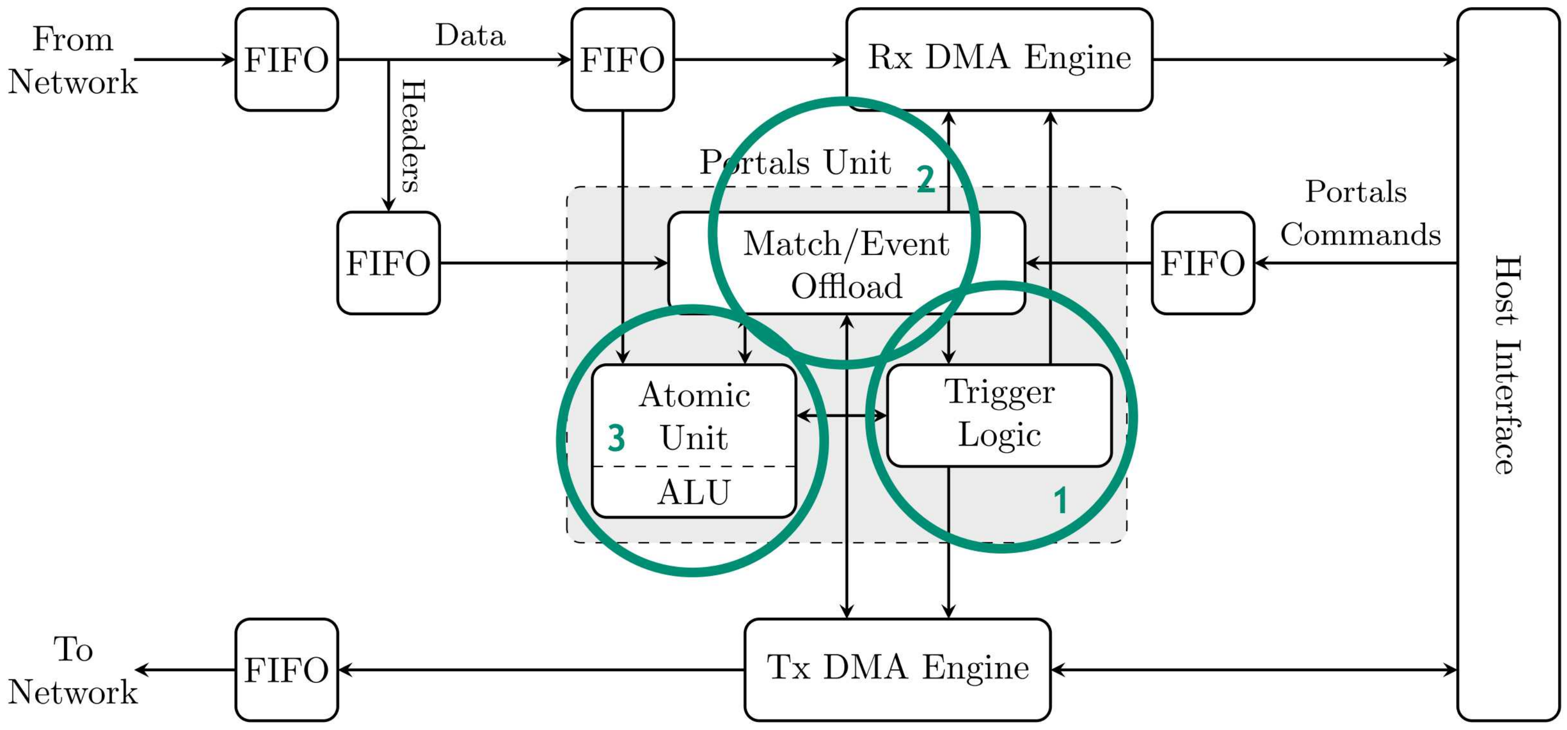
- Traditional offloaded network processing:
  - Often task-specific, even when general-purpose hardware is used (CPUs, FPGAs)
  - May sometimes allow users to offload general-purpose kernels (e.g., Myrinet, SPiN)
  - Nonetheless deadline-based:
    - Compute resources must be available for incoming data
      - Limits on number of instructions that can be executed per message or packet (see next slide)
      - Kernels written for one system may not honor deadlines imposed by future systems
      - Desideratum of harvesting compute resources when the network is idle not addressed
- INCA: Enable deadline-free, general-purpose compute offloading

Schonbein, Grant, Dosnjh, Arnold. "INCA: In-Network Compute Assistance," in Proceedings of SC'19, November 2019.

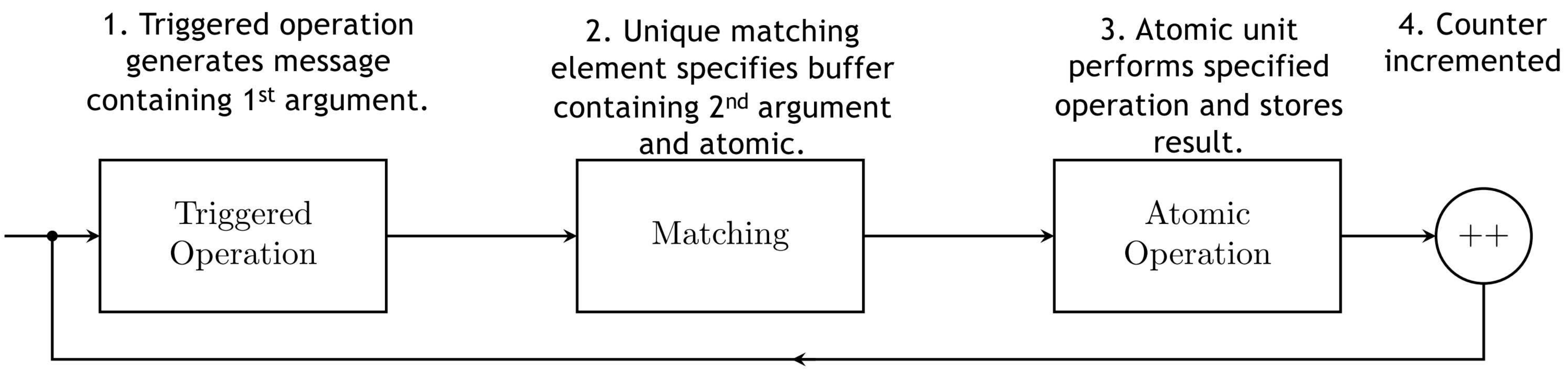
INCA: In-Network Compute Assistance

- Instruction limits
  - For instance: for a data rate of 200Gb/s, 64B packets, and 32 2.5GHz cores with 1 IPC, a stream-based approach is limited to packet-processing kernels of less than 500 instructions.
  - This limit can exclude:
    - Dot product, matrix transpositions, and Hadamard products involving 1024 elements or more.
    - Significant matrix multiplications (exceeding 16 64bit elements).
    - Simple 3x3 convolutions
    - Linear interpolations exceeding 16 64bit points.

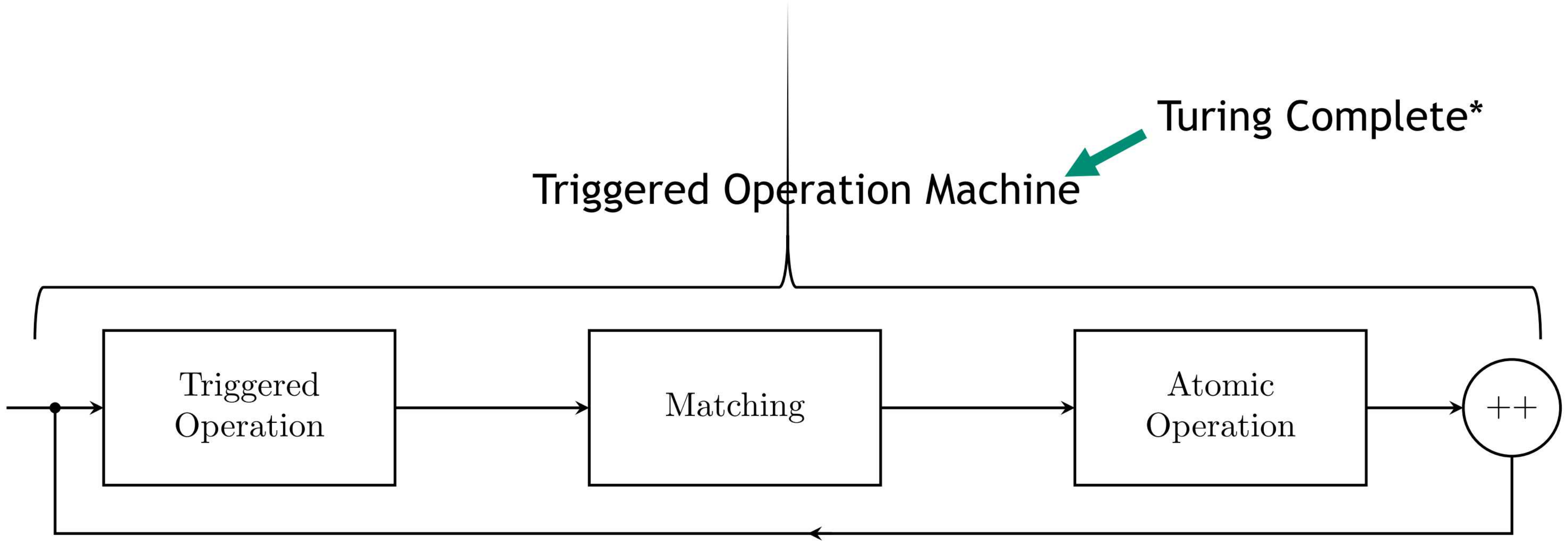
INCA: In-Network Compute Assistance



INCA: In-Network Compute Assistance


1. Triggered operation generates message containing 1st argument.
2. Unique matching element specifies buffer containing 2nd argument and atomic.
3. Atomic unit performs specified operation and stores result.
4. Counter incremented
Triggered Operation
Matching
Atomic Operation
++

INCA: In-Network Compute Assistance



*assuming some slight modifications to existing hardware…

INCA: In-Network Compute Assistance

- Kernel execution is deadline-free:
  - Execution can be suspended (e.g., by incoming network traffic) between triggered messages being generated and matching.
- Kernels are forward compatible:
  - No limits on number of instructions, so faster networks do not imply fewer instructions can be executed.
  - … and faster networks mean INCA kernels will execute faster when they are scheduled.
- Kernel execution can continue when the network is otherwise idle.

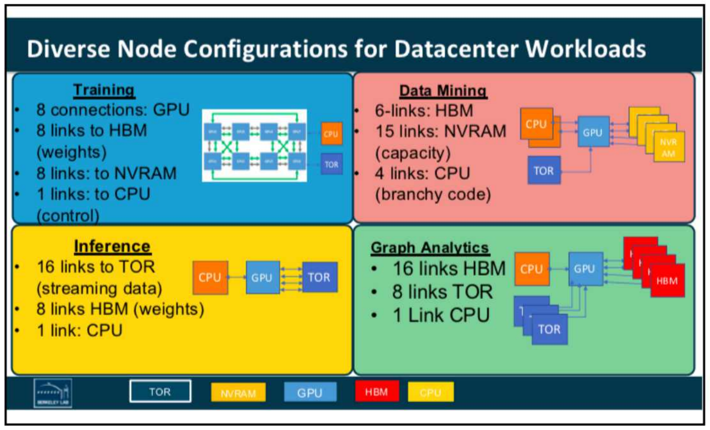
INCA: In-Network Compute Assistance

- INCA: abstract triggered operation machine programs into something more recognizable, e.g.:

Algorithm 2 INCA Dot Product

```
1:  PUTL i, 0, i16
2:  PUTL $r_0$, i, i16
3:  LT $r_0$, $r_0$, b, i16
4:  BLEZ $r_0$, 10
5:  PUTL $r_1$, A[i], f
6:  MUL $r_1$, $r_1$, B[i], f
7:  ADD c, c, $r_1$, f
8:  ADD i, i, 1, i16
9:  JMP 2
10: END
```

Disaggregated Systems Architecture



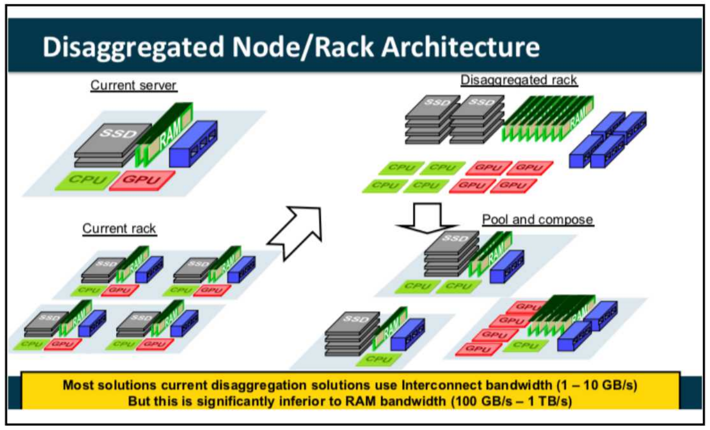
Diverse Node Configurations for Datacenter Workloads

Training
- 8 connections: GPU
- 8 links to HBM (weights)
- 8 links: to NVRAM
- 1 links: to CPU (control)

Data Mining
- 6-links: HBM
- 15 links: NVRAM (capacity)
- 4 links: CPU (branchy code)

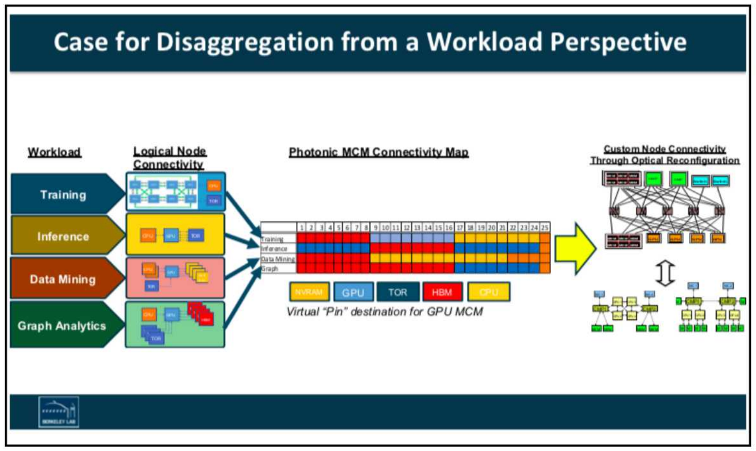
Inference
- 16 links to TOR (streaming data)
- 8 links HBM (weights)
- 1 link: CPU

Graph Analytics
- 16 links HBM
- 8 links TOR
- 1 Link CPU



Disaggregated Node/Rack Architecture

Most solutions current disaggregation solutions use Interconnect bandwidth (1 – 10 GB/s)
But this is significantly inferior to RAM bandwidth (100 GB/s – 1 TB/s)

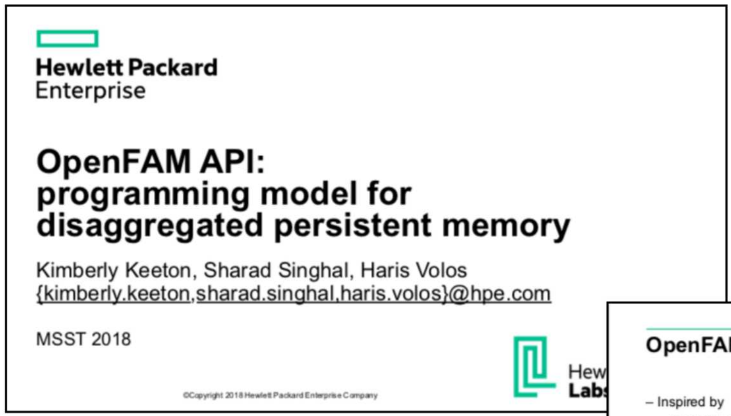


Case for Disaggregation from a Workload Perspective

Workload
Logical Node Connectivity
Photonic MCM Connectivity Map
Custom Node Connectivity Through Optical Reconfiguration
Training
Inference
Data Mining
Graph Analytics
NVRAM
GPU
TOR
HBM
CPU
Virtual "Pin" destination for GPU MCM

*Stolen from John Shalf

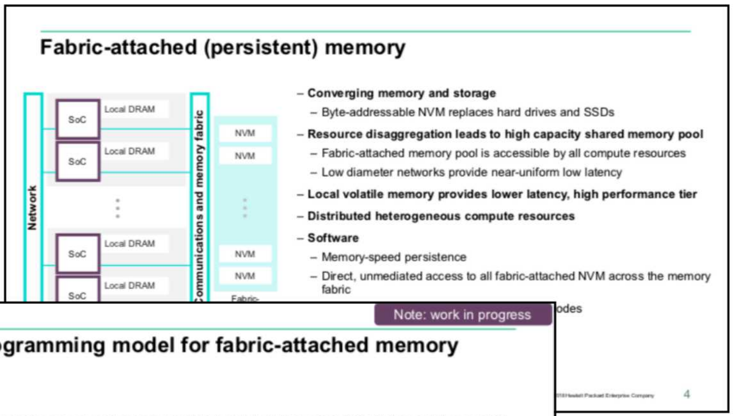
HPE's OpenFAM – Disaggregated Persistent Memory


Hewlett Packard Enterprise

OpenFAM API: programming model for disaggregated persistent memory

Kimberly Keeton, Sharad Singhal, Haris Volos
{kimberly.keeton,sharad.singhal,haris.volos}@hpe.com

MSST 2018

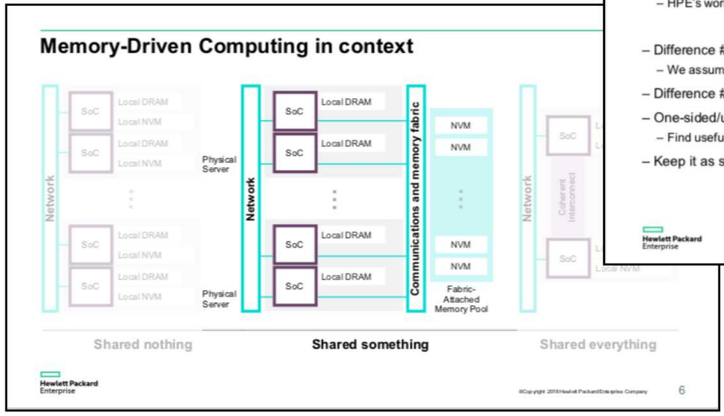

Fabric-attached (persistent) memory

Network
SoC
Local DRAM
Communications and memory fabric
NVM
Fabric-

- **Converging memory and storage**
  - Byte-addressable NVM replaces hard drives and SSDs
- **Resource disaggregation leads to high capacity shared memory pool**
  - Fabric-attached memory pool is accessible by all compute resources
  - Low diameter networks provide near-uniform low latency
- **Local volatile memory provides lower latency, high performance tier**
- **Distributed heterogeneous compute resources**
- **Software**
  - Memory-speed persistence
  - Direct, unmediated access to all fabric-attached NVM across the memory fabric

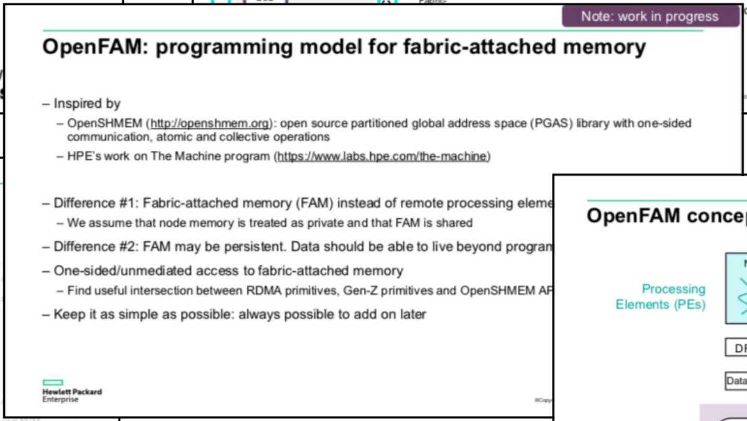
odes

4


Memory-Driven Computing in context

Hewlett Packard Enterprise

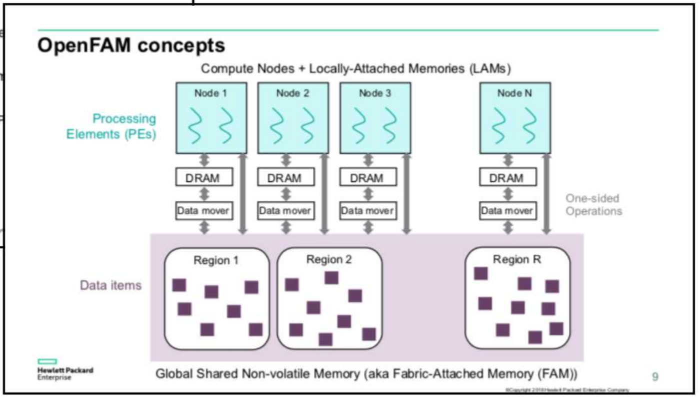
6


Note: work in progress

OpenFAM: programming model for fabric-attached memory

- Inspired by
  - OpenSHMEM (http://openshmem.org): open source partitioned global address space (PGAS) library with one-sided communication, atomic and collective operations
  - HPE's work on The Machine program (https://www.labs.hpe.com/the-machine)
- Difference #1: Fabric-attached memory (FAM) instead of remote processing eleme
  - We assume that node memory is treated as private and that FAM is shared
- Difference #2: FAM may be persistent. Data should be able to live beyond progran
- One-sided/unmediated access to fabric-attached memory
  - Find useful intersection between RDMA primitives, Gen-Z primitives and OpenSHMEM AF
- Keep it as simple as possible: always possible to add on later

Hewlett Packard Enterprise


OpenFAM concepts

Compute Nodes + Locally-Attached Memories (LAMs)
Node 1
Node 2
Node 3
Node N
Processing Elements (PEs)
DRAM
Data mover
One-sided Operations
Data items
Region 1
Region 2
Region R
Global Shared Non-volatile Memory (aka Fabric-Attached Memory (FAM))

Hewlett Packard Enterprise

9

Summary

- Innovation and technology are driven more by economics than science
- SmartNICs are fundamental enablers for Cloud Computing
  - Not clear that they will enable HPC-style offload
  - Disaggregated system architecture may eliminate need for virtualization

Questions?