

*Exceptional service in the national interest*



## Hardware/Software Co-Design for High Performance Interconnects for Extreme-Scale Systems

Ron Brightwell, R&D Manager  
Center for Computing Research



Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

# Portals Interconnect Programming Interface

- Developed primarily by Sandia, U. New Mexico, Intel
- Deployed on several production massively parallel processing (MPP) and cluster systems
  - 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
- At least three hardware implementations currently under development
- Portals influence can be seen in InfiniBand APIs and libfabric (Intel & others)



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

# 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 scalability in stable, high-performance computing systems. Paragon SP70 supercomputers provide the overall capacity, balanced performance and cost-of-ownership 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/R 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.

**High Performance**

- Configuration of 64 to 24 GFLOPS, 0.35 to 21 KMIPS
- Broadcast node-to-node communication at up to 175 MB/sec, full duplex per channel
- Aggregate hardware interconnect location bandwidth to 5.4 Gflop
- Scalable main memory to 17 GB\*
- Scalable internal disk storage to 272 GB

**Production Ready**

- Industry-standard (OS/390) operating system
- Multiple HPPC, EBCnet, FDDC connections
- Simultaneous batch and interactive operation
- Resource control, accounting and scheduling tools
- Designed for high availability
- 24x7x365 operation

**Enhanced Programmability**

- Fortran 77, C, Ada
- High Performance Fortran™, C++™
- Portable interactive, graphical development environment
- Performance-optimized node libraries and parallel equation solvers
- X Windows System™, DGL™, OpenGL™ and OS/2™ graphics
- Compatibility with Intel i860™ family

**System Overview**

The Paragon supercomputer is a distributed memory multiprocessor based on Intel's Intel i860 architecture, which has been developed over the course of generations of parallel supercomputers and is implemented using Intel's advanced microprocessor and



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 peak at 1.2 GB/sec.

General-purpose GP nodes have a single Intel i860 XP application processor as well as an additional Intel 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 (HPPI) or Small Computer System Interface (SCSI). The nodes can be configured with 16, 32, 96 or 192 MB of high speed, error correcting dynamic RAM, accessible at 400 MB/sec. The 46-bit memory unit provides interleave, dual bank memory accesses.

**Interconnect Architecture**


Along with node performance, inter-node communications is a determining factor for sustained multi-processor performance. The Paragon system's interconnect network provides high bandwidth, low-latency communications and links programmers from having to concern themselves with interconnect topology. All nodes appear to be connected to all other nodes, and communication performance is uniform. Broadcast bandwidth, a measure of overall system capacity, scales from 0.7 GB for a Paragon Model XP/R 16 to 5.4 GB for a Paragon SP70 Model 24.

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 message, the node's Intel 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).

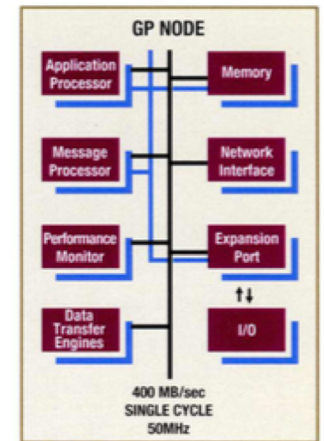
**Message Routing.** The actual transmission of messages is carried out by an independent routing system of custom-designed Mesh Router Controller (MRC) is one for each node, arranged in a two-dimensional mesh. These fixed-function MRCs have a bandwidth of up to 175 MB/sec and can receive any peripheral device, high speed or low speed, or local area network. Hardware is also time to set up the transfer of the first byte of data is as low as 40 nsec per MRC transfer. The physical location of nodes becomes unimportant for performance. Like the Intel 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.



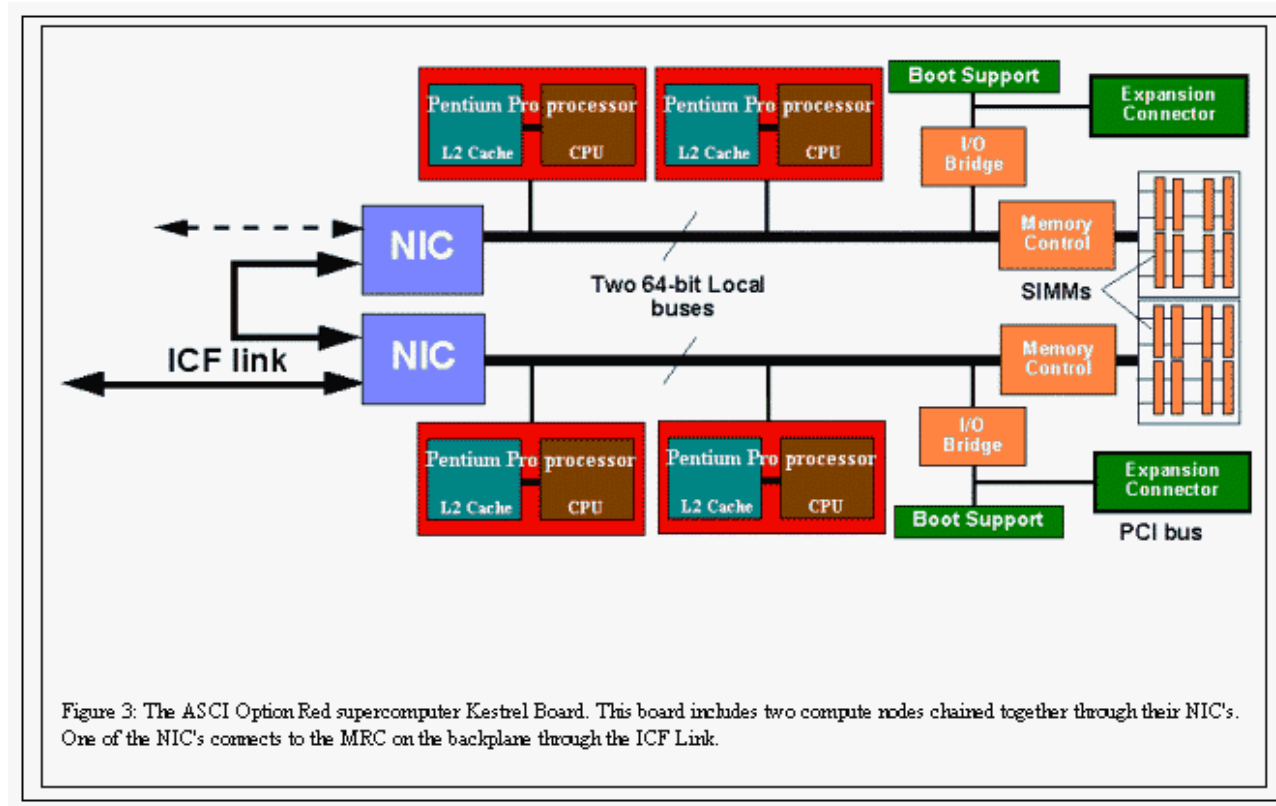
**Message Processor.** When an application decides to send a message, the node's i860 XP message processor handles message-protocol processing and frees the application processor to continue with numeric computing. Messaging software is executed from the message processor's internal cache, enabling overlapped 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



# 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 MPPs

- 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

# Design Philosophy: Don't Arbitrarily Constrain

- Connectionless
  - Easy to do connections over connectionless
  - Impossible to do vice-versa
- One-sided
  - Easy to do two-sided over one-sided
  - Hard to do vice-versa
- Matching
  - Needed to enable flexible independent progress
  - Otherwise matching and progress must be done by upper layers
- Offload
  - Straightforward to onload API designed for offload
  - Hard to do vice-versa (see TCP Offload Engines [TOE])
- Progress
  - Must be implicit



# Kernel-Level Networking

- UNIX IP sockets (UDP/IP, TCP/IP)
- Kernel contains a ring of send and receive buffers
- Send
  - Application calls *write()* system call
  - Kernel copies data from user-space into kernel buffer or network device memory
  - System call returns number of bytes sent
- Receive
  - Network device interrupts OS when data arrives
  - Kernel copies data from network device into kernel buffers
  - Kernel copies data from kernel buffers to user memory during *read()*
  - System call returns number of bytes received
- Checking for incoming data
  - Use *select()* or *poll()* system calls to see if data can be read (or written)
  - Use *sigaction()/fcntl()* to receive a SIGIO signal when data can be read

# User-Level Networking

- Network device is directly controlled by application process after initial setup
- Send
  - Application process writes a command to the network device
  - Network device copies data directly from user-space onto the network
- Receive
  - Application process provides buffer(s) to the network device before data arrives
  - Network device copies data directly from network into user-space
- Checking for incoming data
  - Poll memory location (application memory or network device memory)
  - No mechanism for OS-generated signals
- Significant performance advantage over kernel-level approach
  - Increases bandwidth by eliminating memory copies
  - Decreases latency by avoiding system calls
  - Provides the opportunity to overlap data movement with computation
- Must coordinate with virtual memory system (page pinning)

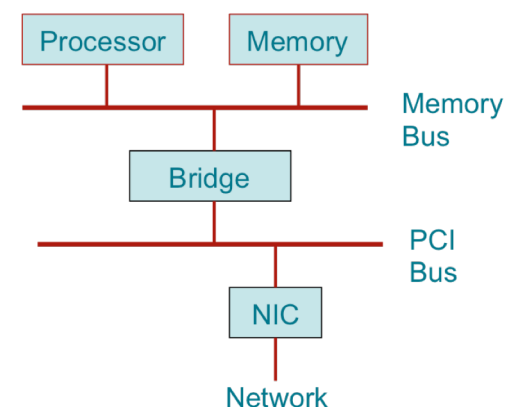
# Programmable User-Level Networks Enabled API Exploration

## ■ 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 QSNNet (~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



# 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)**

- 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

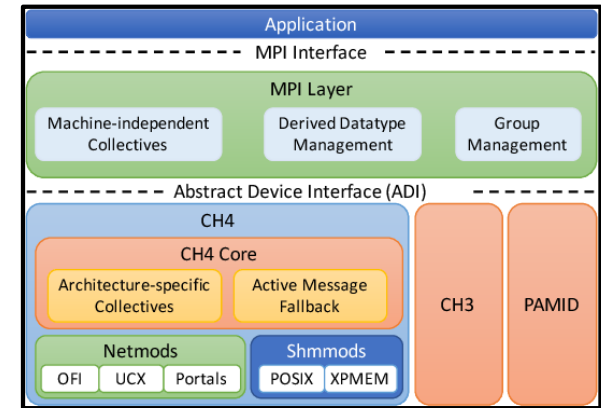
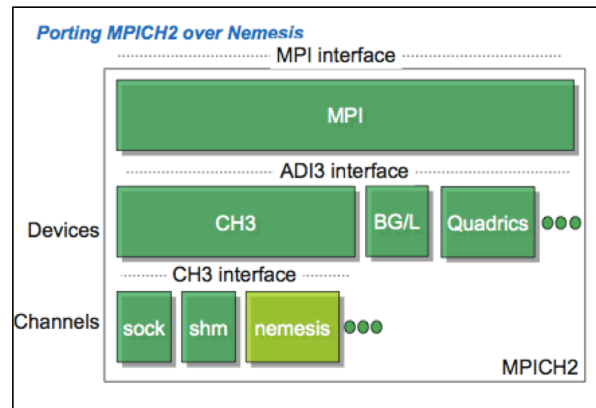
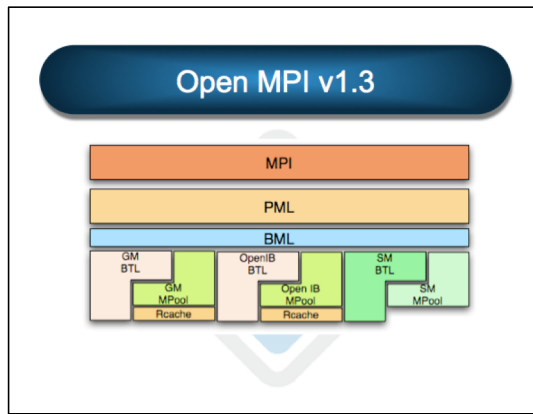
## **MPI Point-to-Point**

- 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

# How to Implement MPI over RDMA (2002-2008)

- Mvapi-ch-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand, Int'l Conference on Parallel and Distributed Computing, Miami, FL, Apr. 2008
- Designing Passive Synchronization for MPI-2 One-Sided Communication to Maximize Overlap, Int'l Conference on Parallel and Distributed Computing, Miami, FL, Apr. 2008
- MPI-2 One Sided Usage and Implementation for Read Modify Write operations: A case study with HPCC, EuroPVM/MPI 2007, Sept. 2007.
- Zero-Copy Protocol for MPI using InfiniBand Unreliable Datagram, IEEE International Conference on Cluster Computing (Cluster'07), Austin, TX, September 2007.
- High Performance MPI over iWARP: Early Experiences, Int'l Conference on Parallel Processing, XiAn, China, September 2007.
- High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters, 21st Int'l ACM Conference on Supercomputing, June 2007.
- Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach, Int'l Symposium on Cluster Computing and the Grid (CCGrid), Rio de Janeiro - Brazil, May 2007
- Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective, Int'l Symposium on Cluster Computing and the Grid (CCGrid), Rio de Janeiro - Brazil, May 2007
- High Performance MPI on IBM 12x InfiniBand Architecture, International Workshop on High-Level Parallel Programming Models and Supportive Environments (HiPS), held in conjunction with IPDPS '07, March 2007.
- High-Performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-Depth Performance Analysis, SuperComputing (SC 06), November, 2006.
- Efficient Shared Memory and RDMA based design for MPI\_Allgather over InfiniBand, EuroPVM/MPI, September 2006.
- Memory Scalability Evaluation of the Next-Generation Intel Bensley Platform with InfiniBand, Hot Interconnect (HOTI 06), August, 2006.
- MPI over uDAPL: Can High Performance and Portability Exist Across Architectures?, Int'l Symposium on Cluster Computing and the Grid (CCGrid), Singapore, May 2006.
- Shared Receive Queue based Scalable MPI Design for InfiniBand Clusters, Int'l Parallel and Distributed Processing Symposium (IPDPS '06), April 2006, Rhode Island, Greece.
- Adaptive Connection Management for Scalable MPI over InfiniBand, International Parallel and Distributed Processing Symposium
- Efficient SMP-Aware MPI-Level Broadcast over InfiniBand's Hardware Multicast, Communication Architecture for Clusters (CAC) Workshop, to be held in conjunction with Int'l Parallel and Distributed Processing Symposium (IPDPS '06), April 2006, Rhode Island, Greece.
- RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits, Symposium on Principles and Practice of Parallel Programming,
- High Performance RDMA Based All-to-all Broadcast for InfiniBand Clusters, International Conference on High Performance Computing (HiPC 2005)
- Supporting MPI-2 One Sided Communication on Multi-Rail InfiniBand Clusters: Design Challenges and Performance Benefits, International Conference on High Performance Computing (HiPC 2005), December 18-21, 2005, Goa, India.
- Designing a Portable MPI-2 over Modern Interconnects Using uDAPL Interface, EuroPVM/MPI 2005, Sept. 2005.
- Efficient Hardware Multicast Group Management for Multiple MPI Communicators over InfiniBand, EuroPVM/MPI 2005, Sept. 2005.
- Design Alternatives and Performance Trade-offs for Implementing MPI-2 over InfiniBand, EuroPVM/MPI 2005, Sept. 2005.
- Can Memory-Less Network Adapters Benefit Next-Generation InfiniBand Systems?, Hot Interconnect (HOTI 05), August, 2005.
- Analysis of Design Considerations for Optimizing Multi-Channel MPI over InfiniBand, Workshop on Communication Architecture on Clusters
- Scheduling of MPI-2 One Sided Operations over InfiniBand, Workshop on Communication Architecture on Clusters (CAC 05) in conjunction with International Parallel and Distributed Processing Symposium
- Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation. SuperComputing Conference, Nov 6-12, 2004, Pittsburgh, Pennsylvania.
- Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms, IEEE Cluster Computing 2004, Sept. 20-23 2004, San Diego, California.
- Zero-Copy MPI Derived Datatype Communication over InfiniBand, EuroPVM/MPI 2004, Sept. 19-22 2004, Budapest, Hungary.
- Efficient Implementation of MPI-2 Passive One-Sided Communication on InfiniBand Clusters, EuroPVM/MPI 2004, Sept. 19-22 2004, Budapest, Hungary.
- Efficient and Scalable All-to-All Exchange for InfiniBand-based Clusters. International Conference on Parallel Processing (ICPP-04), Aug. 15-18, 2004, Montreal, Quebec, Canada.
- Design and Implementation of MPICH2 over InfiniBand with RDMA Support. Int'l Parallel and Distributed Processing Symposium (IPDPS 04), April, 2004.
- Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. Int'l Parallel and Distributed Processing Symposium (IPDPS 04), April, 2004.
- High Performance Implementation of MPI Datatype Communication over InfiniBand. Int'l Parallel and Distributed Processing Symposium (IPDPS 04), April, 2004.
- Implementing Efficient and Scalable Flow Control Schemes in MPI over InfiniBand. Workshop on Communication Architecture for Clusters (CAC 04)
- High Performance MPI-2 One-Sided Communication over InfiniBand. IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 04), April, 2004.
- Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters. Euro PVM/MPI Conference, September 29-Oct 2, 2003, Venice, Italy.
- High Performance RDMA-Based MPI Implementation over InfiniBand. 17th Annual ACM International Conference on Supercomputing, San Francisco Bay Area. June, 2003.
- Impact of On-Demand Connection Management in MPI over VIA, Cluster '02, Sept. 2002.

# Network Portability Abstraction Layers Abound



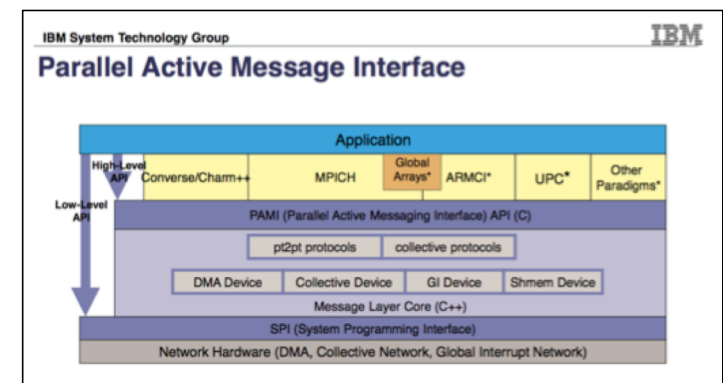
**Goals**

- Provide a common low-level scalable, robust, portable, simple and performance driven communication API for multiple parallel programming models over modern network interfaces
- Increasing code reusability and reducing development effort
- Include performance/power measurement capabilities in a central location

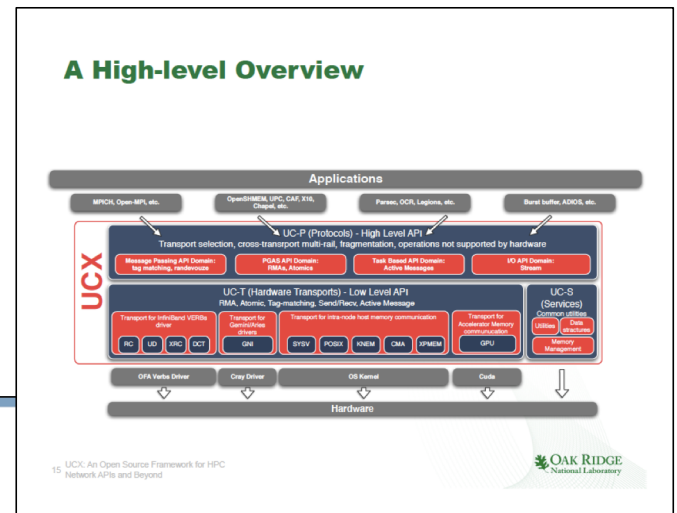
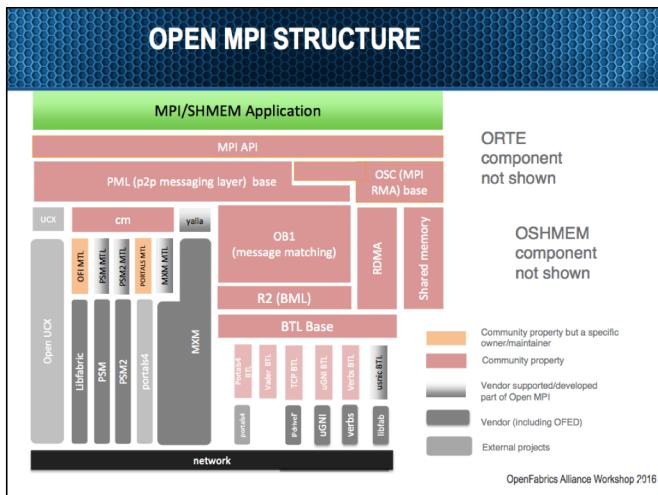
The UCCS (Unified Communication Core) diagram shows a central core connected to various network interfaces and programming models. The interfaces include OpenSHMEM, X10, MDHIM, UPC, Chapel, IO Lib, MPI, BlueGene, InfiniBand, Ethernet, SeaStar, Gemini, Shared Memory, and hwarp.

**GASNet Design Overview: System Architecture**

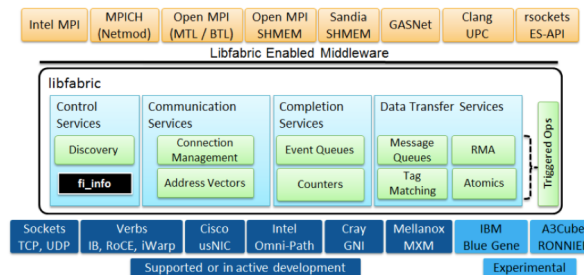
- Two-Level architecture is mechanism for portability
- GASNet Core API
  - Most basic required primitives, narrow and general
  - Implemented directly on each network
  - Based on Active Messages lightweight RPC paradigm
- GASNet Extended API
  - Wider interface that includes higher-level operations
    - puts and gets w/ flexible sync, split-phase barriers, collective operations, etc
  - Have reference implementation of the extended API in terms of the core API
  - Directly implement selected subset of interface for performance
    - leverage hardware support for higher-level operations



# Recent Efforts to Develop Lower-Level Transport APIs



## OFI - Libfabric



# 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
  - Big Data
- Desire to reduce development costs
  - Provide one network abstraction for all ULPs
  - Large porting effort is a strong indication of the semantic mismatch
- “Thin is in”
  - Optimize to a semantic mismatch
  - Get as close to the functionality you don’t really want as possible
    - Communication as well as memory management (page pinning)
- Vendor differentiation
  - Which really defeats the portability goal

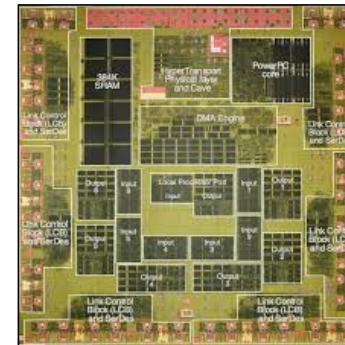


# Some Fundamental Principles

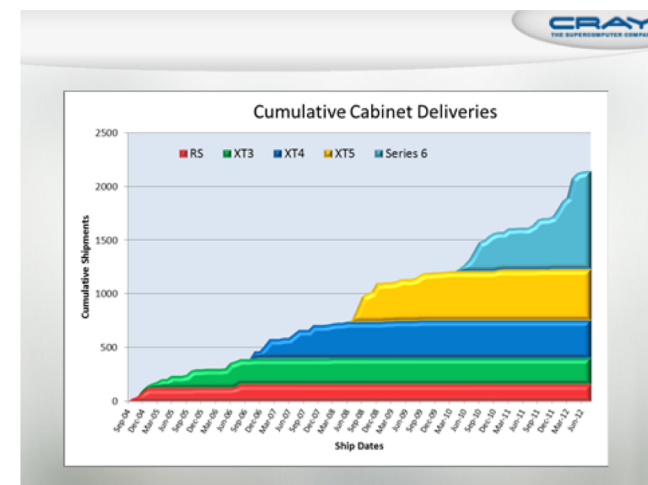
- More layers of software degrades performance (and scalability)
- Hardware almost always outperforms software
- Software fixes to hardware are usually really slow

# Red Storm – Prototype for Cray XT Series

- Architected by Sandia, engineered jointly with Cray
  - 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



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

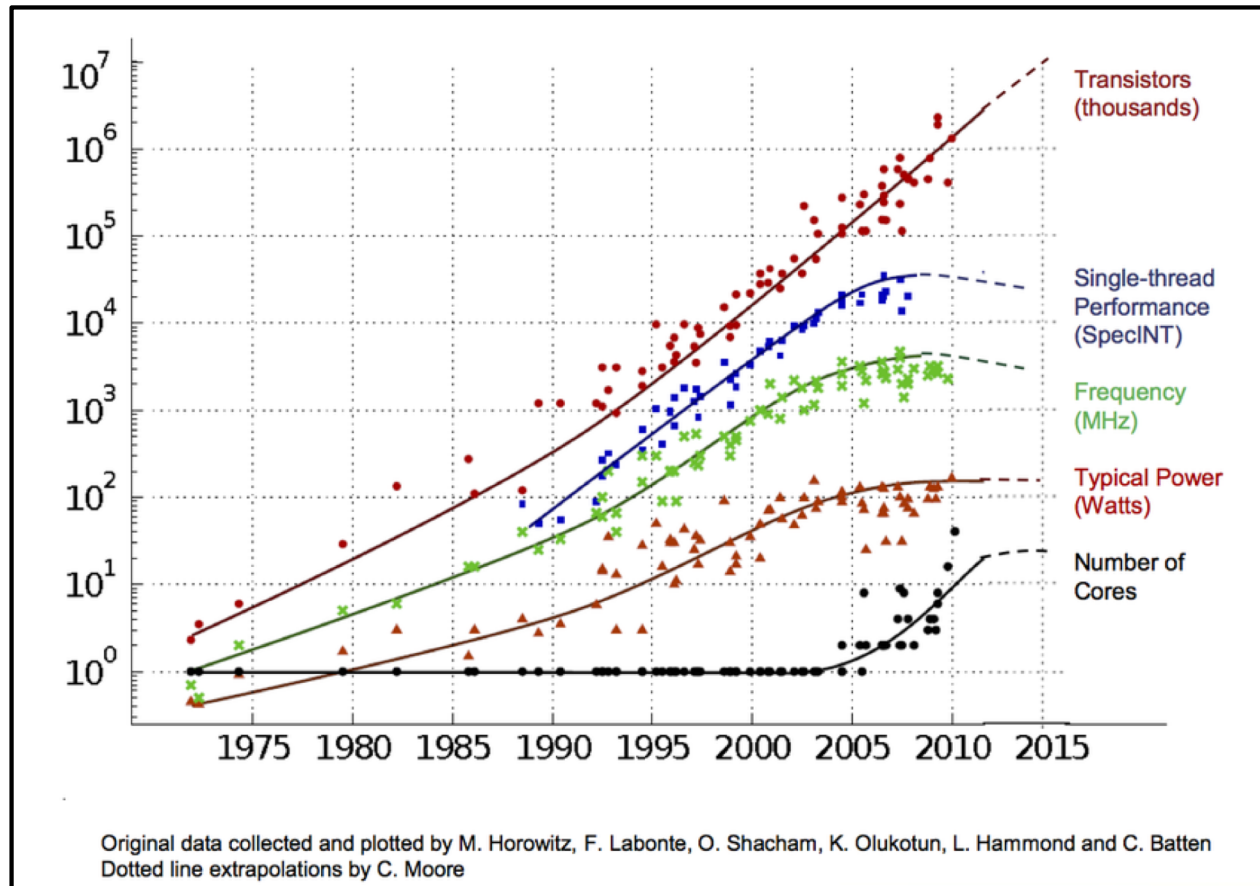


\*Source: IDC #209251 *Technical Computing Systems: Competitive Analysis*, November 2007

# Onload Versus Offload Argument (~2005)

- Why design a custom NIC for offload?
- Just dedicate a core
  - A 3 GHz Xeon will outperform a 500 MHz embedded processor on network protocol processing
  - A custom ASIC is way too expensive (especially for the small HPC market)
- Cost will go down as core count increases
- Cores won't be getting slower, right?

# Core Clock Frequency Stalled in ~2007



# Cray Core Specialization

- Dedicate “OS” cores to handle MPI progress
  - MPI progress threads run on a dedicated set of cores

PROCEEDINGS OF THE CRAY USER GROUP 2012

## 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 nodes available on Cray XE compute nodes. With CoreSpec, host cores on a node are dedicated to running the parallel application while one or more cores are reserved for OS and service threads. The MPI/CLC MPI implementation has been enhanced to make use of the 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 general-ized 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: hpritch@pritchardlab.usf.edu; roweth@cray.com

This material is based upon work supported by the Defense Advanced Research Projects Agency under the Agreement No. FA9550-02-1-0188-0001. 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-AC02-04OR21400.

## 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

# Portals 4 Reference Implementation

- OpenFabrics Verbs, UDP, shared memory transports
- Initial implementation by System Fabric Works
- Provides a high-performance reference implementation for experimentation
- Help identify issues with API, semantics, performance, etc.
- Independent analysis of the specification
- Enables development of ULPs

# Reference implementation issues

- Extra layer of software between ULP and hardware
  - Impacts latency performance
  - We are violating one of our fundamental principles 😊
- Needs a progress thread
  - Impacts latency performance
  - Issues when ULP wants a progress thread too
- Do we modify API for hardware we have or continue to design for hardware we need?
- Portals should be slow or we're not doing our job 😊

# Portals Hardware and Co-Design Activities

## BXI application environment

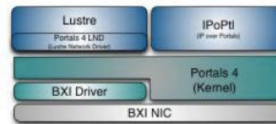
BXI comes with a complete software stack to provide optimal performance and reliability to all traditional HPC components.



BXI Computing stack

- Parallel applications can take full advantage of the capabilities of the BXI network using MPI, SHMEM or UPC communication libraries.
- All components are implemented directly using the Portals 4 API.
- Kernel services are also implemented using the kernel Portals 4 implementation.
- A Portals 4 LND (Lustre Network Driver) provides the Lustre parallel filesystem with a direct / native access to Portals 4.

- The IPOPtI (IP over Portals) component makes it possible to have large scale, efficient and robust IP communication for legacy software.



BXI Kernel services

For more information: Please contact [hpc@atos.net](mailto:hpc@atos.net)

Your business technologists. Powering progress



**Fig. 6. Portals Accelerator Processor (PAC).**

used the untyped functional abstraction level as described in Cai and Gajski [2008] in which the low-level timing details are hidden so as to accelerate the simulation execution time.

The basic components of our novel PAC system are shown in Figure 6. The functionality of each component is analyzed in the following subsections.

**3.2.1. Master/Slave Ports.** PAC is connected to the virtual bus through master and slave ports, depending on whether it initiates or responds to bus transactions. In order to connect PAC to OVP's virtual bus, we utilized both the Innovative CPU Manager (ICM) API for the Global Address spaces and the Peripheral Programming Model (PPM) API for the open, close, read and write operations of the Master and Slave Ports [Jensen et al. 2016]. Specifically, three master ports are handling the read and write of the Portals messages to the other nodes, while the operations of each accelerator are triggered by three distinct OVP events (i.e., one for each incoming message type) using three slave ports.

**3.2.2. Message Buffers.** Three Message Buffers are placed at three corresponding slave ports triggering the accelerator when three different types of incoming Portals message arrive. These buffers store the requests from the processor, which are effectively simple memory writes to appropriate addresses. In this way, the overhead of the host-processor is minimal, since it can return to its normal operations just after it writes the Portals command to the message buffer. The master ports, on the other hand, do not need buffers as the headers and the payloads of the Portals messages are stored in the corresponding global memory spaces. In our case, we use a Message Buffer of size `NPROCS*MAX_MSG_SIZE` bytes; when the message buffer gets full an interrupt is raised to the processor.

**3.2.3. Portals Message Processor.** The Portals Message Processor (PMP) orchestrates the data flow through all PAC's components based on the control flow imposed by each Portals command. Initially, when a request message is placed at the message buffer, PMP decodes the 'incoming' type and identifies the message (Header and possibly Payload) position in the Global Address Spaces. Hence, besides the message buffers, PMP communicates with the master ports that have access to the Global Address Spaces as well as with the Accelerator's buffers. Moreover, PMP issues certain list

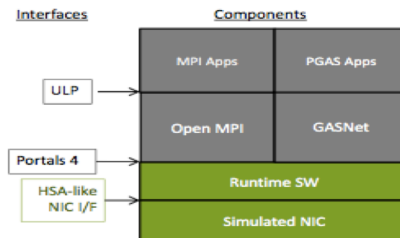
\*OVP PPM is an application modeling the interface and the connections (to ports, and the like) to the

©Atosian Inc. December 2016.

## 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<sup>[6]</sup>
  - System call emulation mode (no OS)
  - AMD GPU model<sup>[7]</sup>
  - Full Support for HSA
  - Tightly coupled system
- Portals 4-based NIC model<sup>[8]</sup>
  - 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
L1-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

[6] N. Binkert, B. Beckmann, S. Black, S. K. Reinhardt, A. Soti, A. Basu, J. Hoehner, D. R. Hoover, T. Krishna, S. Sardashti, R. Sen, S. Sewell, M. Shoshitaishvili, P. D. Stuebner, D. Wang, and A. Wood, "The gem5 simulator," SIGARCH Comput. Archit. News, vol. 7, pp. 1-7, 2011.

[7] AMD, "AMD GPU Accelerator Modeling Framework supports gem5," <https://www.amd.com/en/press-events/press-releases/detail/4074>.

[8] Sandia National Laboratories, "The Portals 4.2.2 network programming interface," <https://www.sandia.gov/Portals/Portals422.html>.



# 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

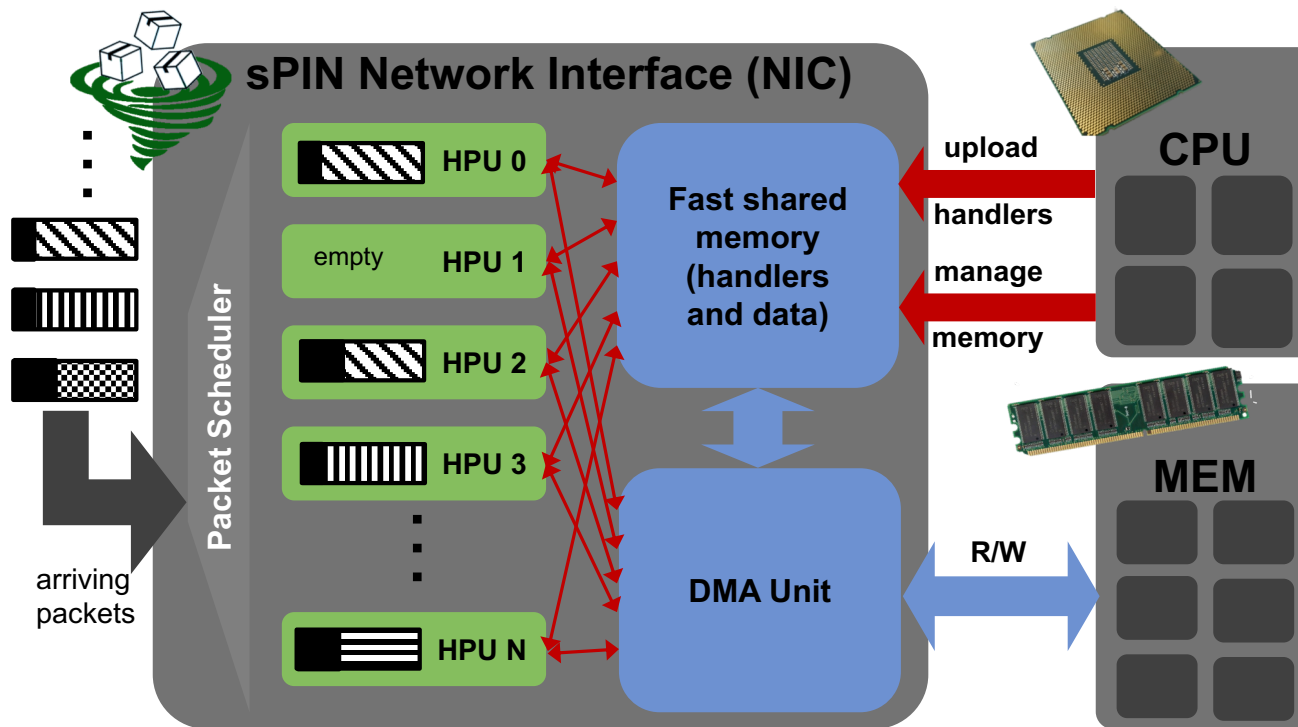
# Issues with Active Messages

- 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)



# 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

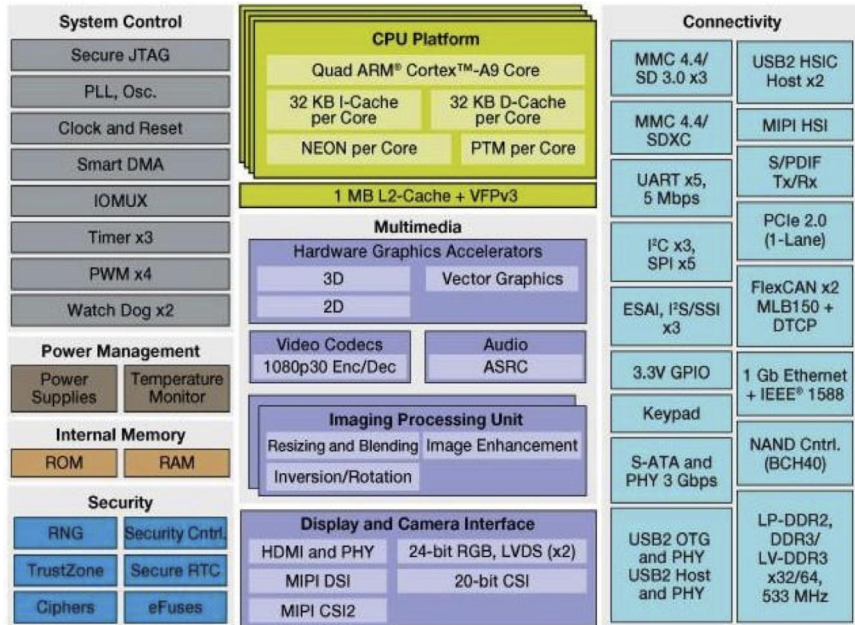
# sPIN Approach

- 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

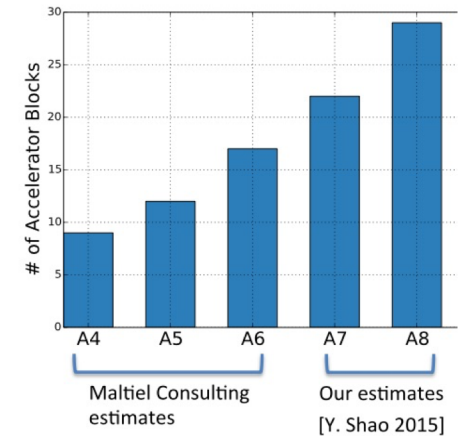
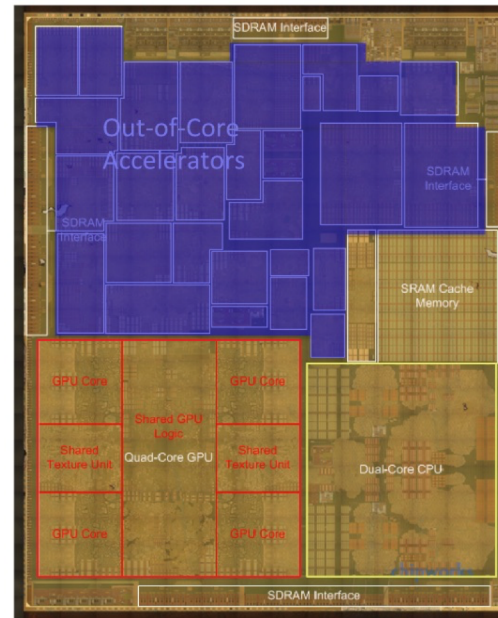
## sPIN Approach (cont'd)

- 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

# Expect More Hardware Specialization in HPC



SmartPhone SoC circa 2016 *Dozens of kinds of integrated HW acceleration*

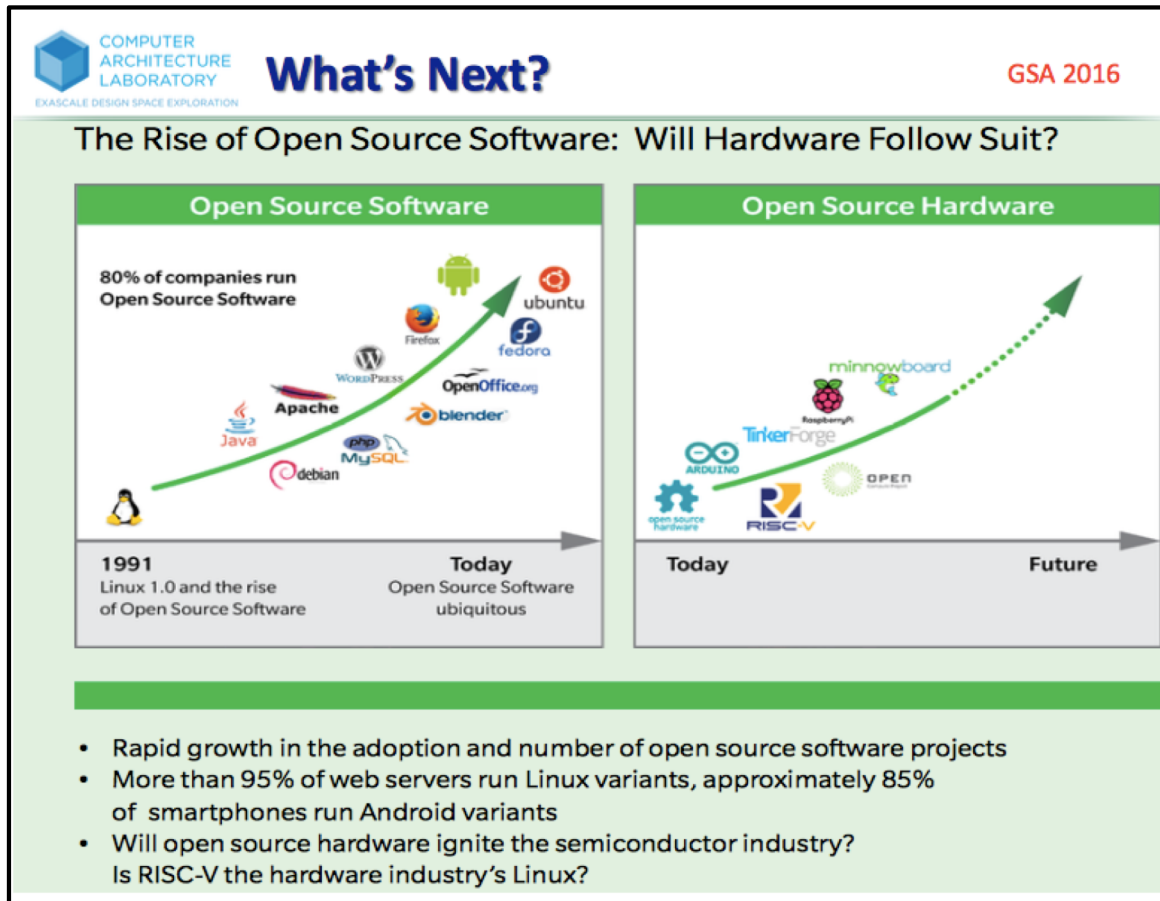


[www.anandtech.com/show/8562/chipworks-a8]

Apple A8 SoC



# Open Source Hardware




# DARPA Effort to Enable Open Source Hardware

Building a Universal Silicon Compiler

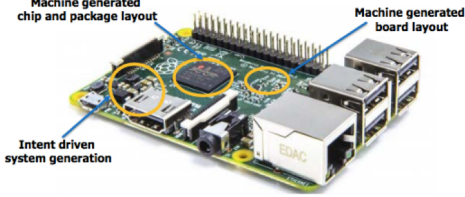
Andreas Olofsson  
Program Manager  
Defense Advanced Research Project Agency (DARPA)

The Salishan Conference on High Speed Computing,  
Glendon Beach, Oregon  
April 24



Distribution Statement "X" (Approved for Public Release, Distribution Unlimited)

**DARPA** IDEA Program Objective



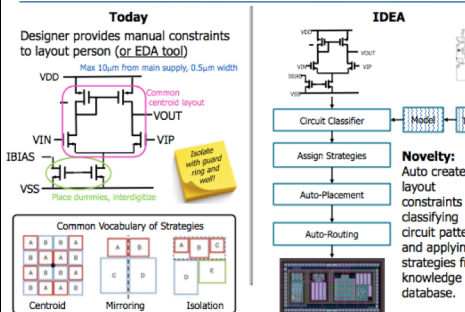
IDEA will create a no-human-in-the-loop hardware compiler for translating source code to layouts of System-On-Chips, System-In-Packages, and Printed Circuit Boards in less than 24 hours

Image source: Raspberry Pi Distribution Statement "X" (Approved for Public Release, Distribution Unlimited) 16

**DARPA** Why now? (how this approach is different)

**Today**  
Designer provides manual constraints to layout person (or EDA tool)

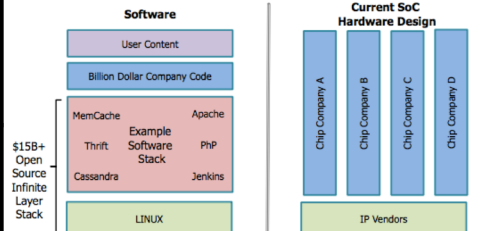
**IDEA**



Centroid	Mirroring	Isolation
----------	-----------	-----------

Distribution Statement "X" (Approved for Public Release, Distribution Unlimited) 17

**DARPA** Reinventing the Chip IP Stack



*POSH will create a viable open source hardware design and verification ecosystem that enables cost effective design of ultra-complex SoCs.*

Distribution Statement "X" (Approved for Public Release, Distribution Unlimited) 18

# Acknowledgments

- Sandia
  - Ryan Grant
  - Scott Hemmert
  - Kevin Pedretti
- ETH Zurich
  - Torsten Hoefler
  - Salvatore Di Girolamo
  - Konstantin Taranov