Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

- Responsible for care for the majority of the US Nuclear Stockpile
- A Serious Enduring Mission
- Not a solved problem space or anywhere close

2019-UCF-UCX-F2F

2019 R&D 100 WINNER

Los Alamos NATIONAL LABORATORY EST.1943

2019-F2F

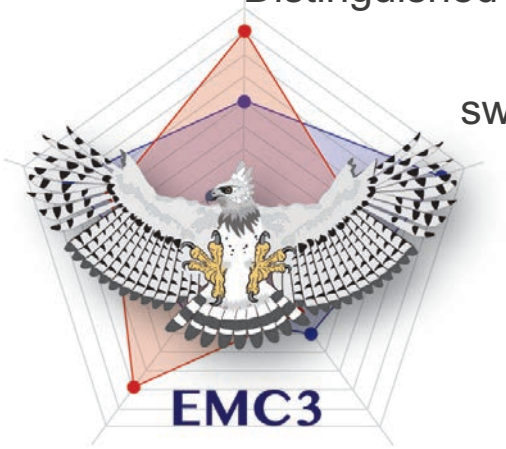
Austin Tx.

Stephen Poole

Distinguished Senior Scientist

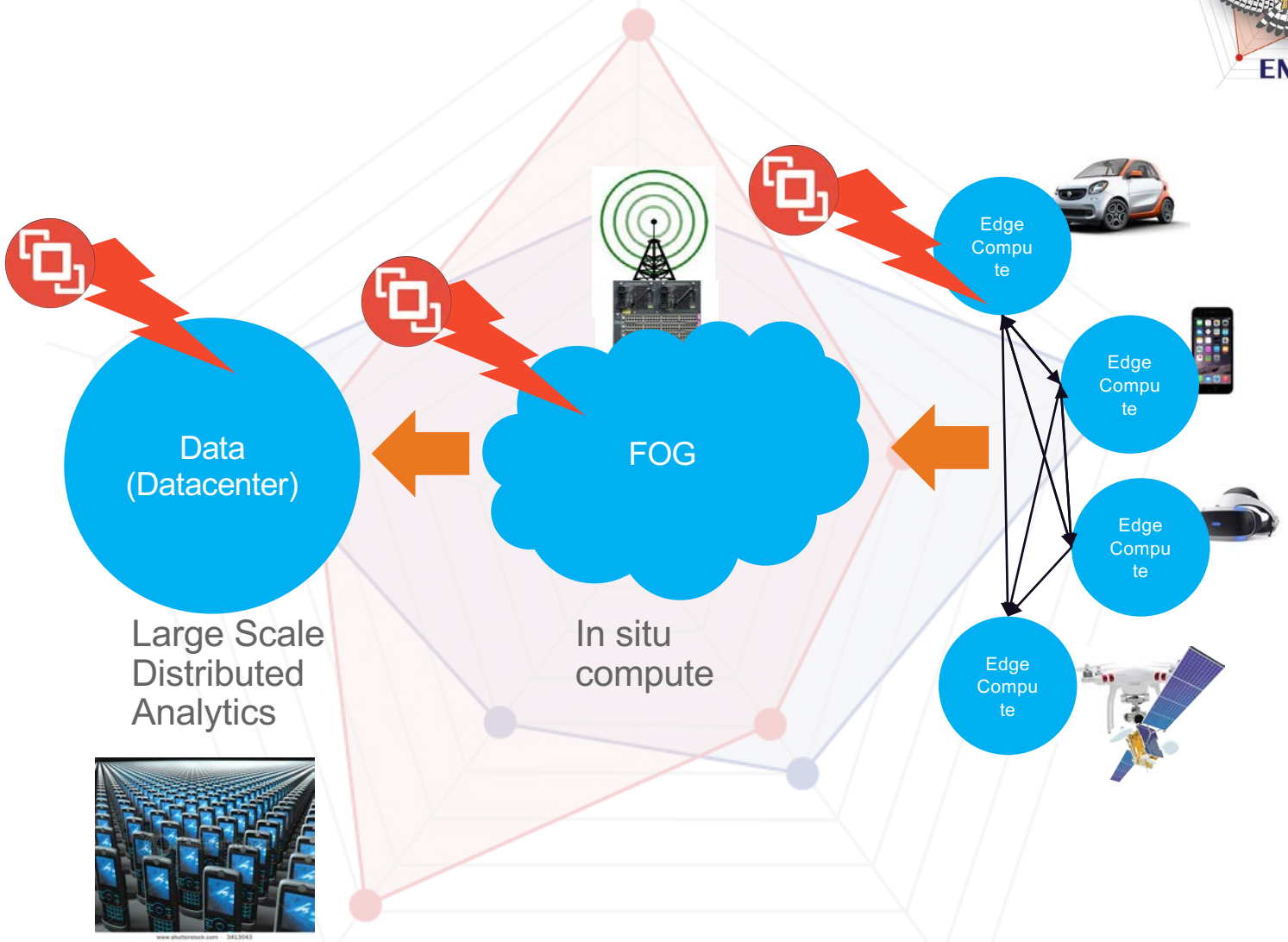
Chief Architect

swpoole@lanl.gov



LA-UR-19-32281

Early idea for graphic



2019 R&D 100 Joint Entry

UCX

Unified Communications X

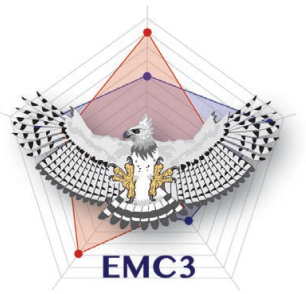
An open-source, exascale-ready communications framework

- Solves decades-old problem in high-performance computing (HPC)
- Frees developers from hardware-specific implementations and laborious porting efforts
- Simplifies deployment of advanced research tools, regardless of system complexity
- Advances fields of artificial intelligence, machine learning, deep learning, and internet of things

Los Alamos NATIONAL LABORATORY EST. 1943

Advanced Micro Devices, Argonne National Laboratory, Arm Ltd., Mellanox Technologies, NVIDIA, Stony Brook University, Oak Ridge National Laboratory, Rice University

Los Alamos has been at the forefront of developing high-speed network interconnects/technologies



- Los Alamos developed one of the first networks, named Hydra, to allow common access to the five CDC machines, 1972
- High-Speed Parallel Interface (HSPI) for inter-computer communication, 50 Mbit/s, 1979-1982
- HIgh-Performance, Parallel Interface (HIPPI), the first gigabit network, 1987
- Gigabyte System Network - GSN 1990's
- Infiniband interconnect came out of ASCI work in the late 1990's
- Optical interconnects started by ASCI ~2000
  - Analysis of optical switches
- QKD – Early 2000's (Spun off)
  - Free space quantum optics
- **In Situ using COTS**
  - Prime candidate for UCX
  - SNIC API (SNAPI)

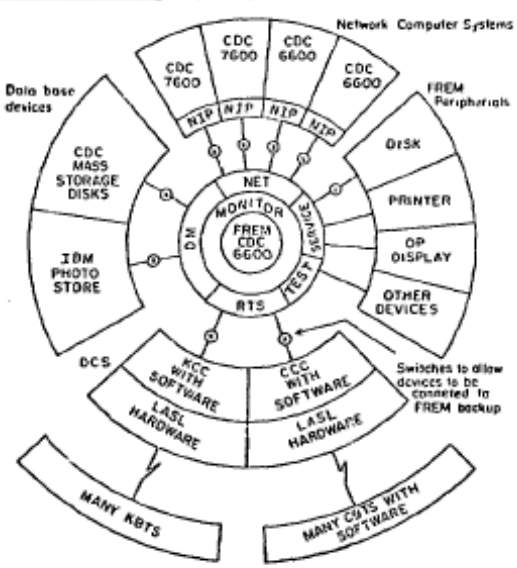
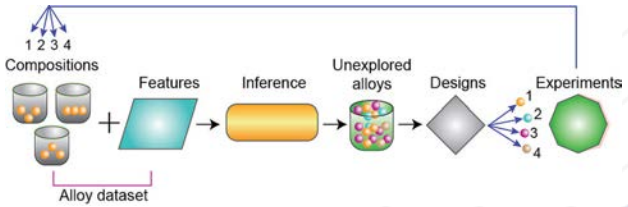


Fig. 1. Hydra system diagram.

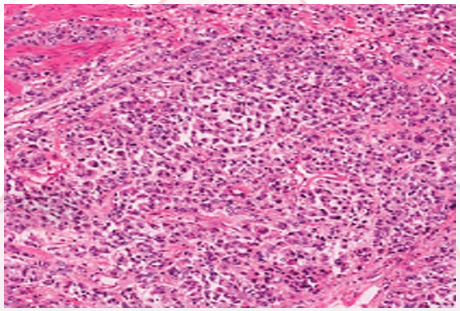
Hydra network design, 1972

High-speed interconnects are mostly a standard commercial off-the-shelf technology now

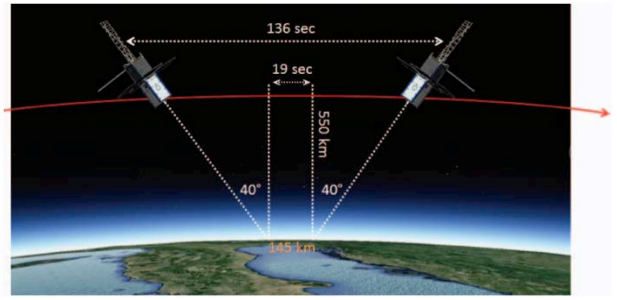
Data analytics is emerging organically and rapidly across many programs



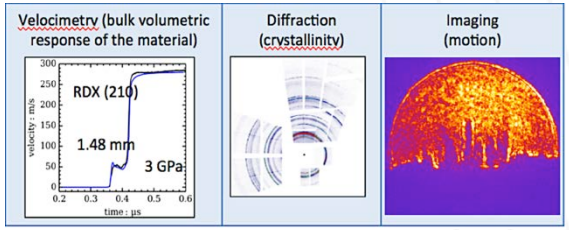
Machine Learning Accelerating Discovery of New Materials, old Materials



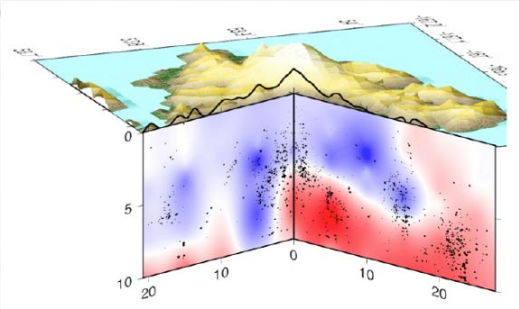
Bioinformatics/ Emergent Diseases



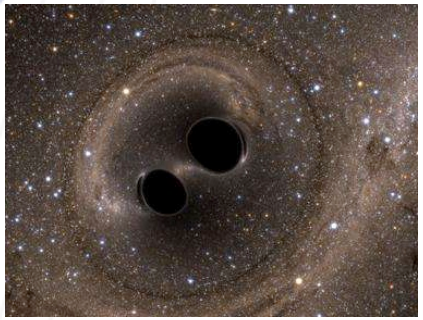
Constellation of CubeSats, Carrying Ultra-Compact Spectral Sensors



Real-time Adaptive Acceleration of Dynamic Experimental Science

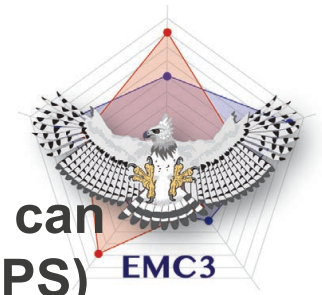


Critical Stress in Subsurface Energy Dynamics

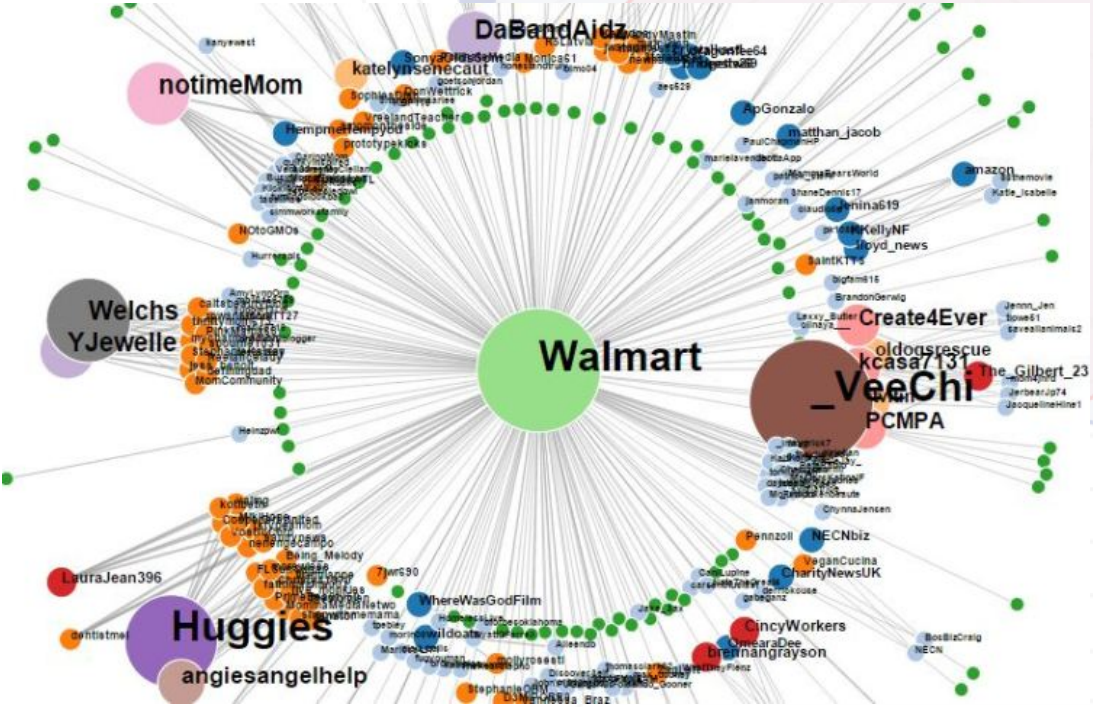


Gravitational Wave Emissions from Colliding Black Holes (LIGO)

Graph Analytics and Event Simulation

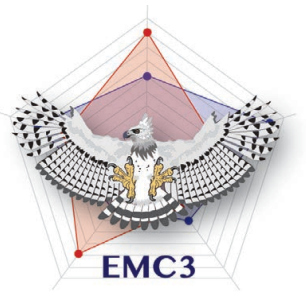


- **Communications intensive, event/data driven, working set can exceed main and often have short words, mostly read. (GUPS)**
  - Worst case is every lookup is not cached, in some random memory location somewhere in the cluster/system
    - Non-cached latency is memory latency ~40-100 cycles
    - Worse is across a low latency network ~1000 cycles (non-deterministic)

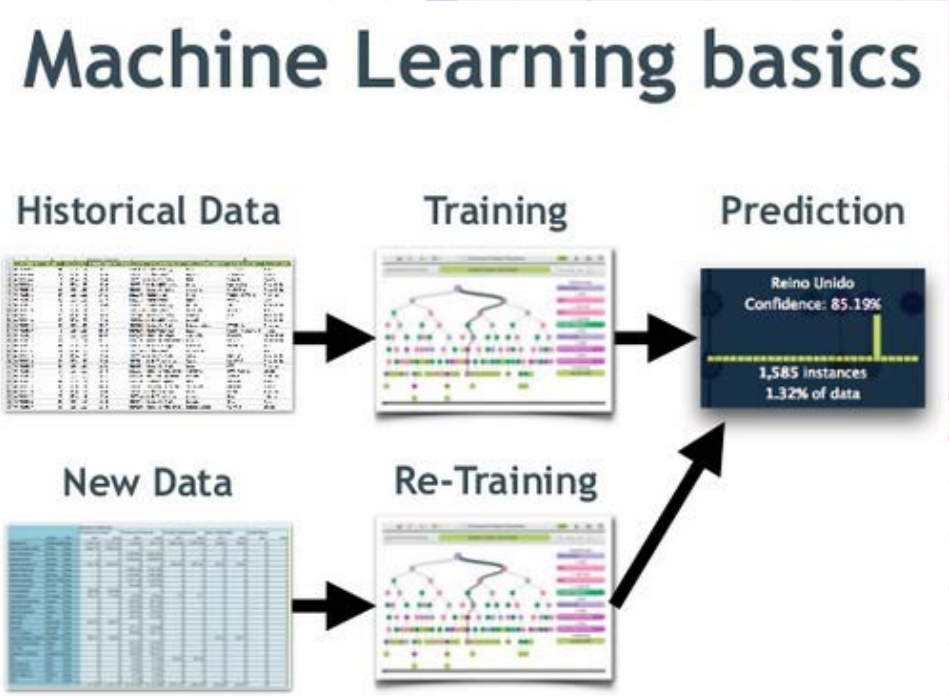


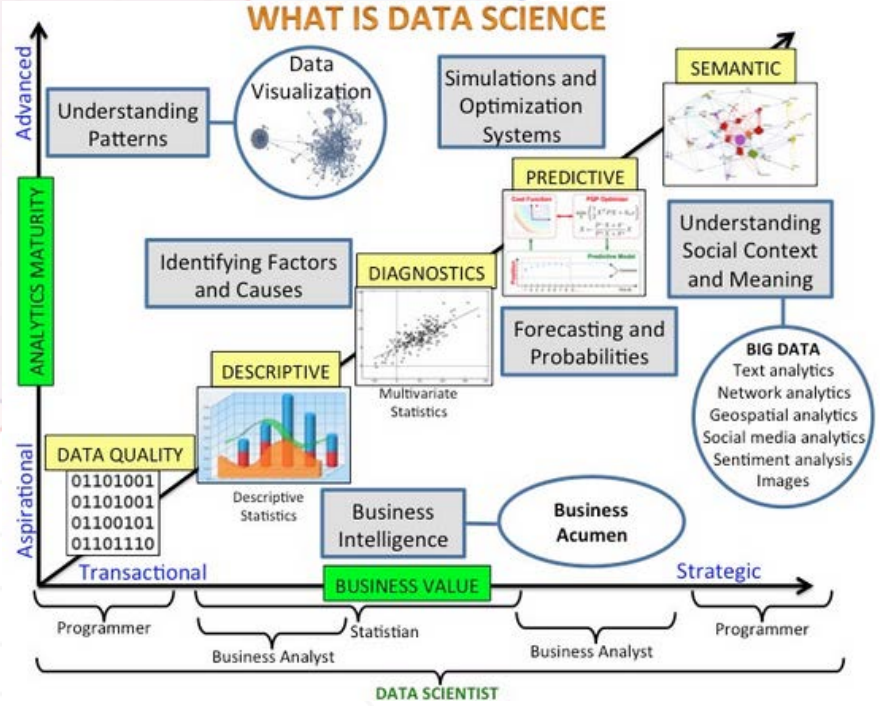
Follow relationships, needles in haystacks, etc.

Many forms of Data Science: Analytics, Streaming Analytics, and some forms of Machine Learning/Deep Learning/AI (potential in situ)



- Often IO Intensive, Data Parallel works very well, often with small word sizes, mostly read dominant
  - Usually parallelizes well
  - IO and Network Bandwidth are sometimes limiters
  - We are working in these areas, so HW has to address issues.





Dated results from Grand Unified File Index has improved by about 5X since this was done

Trinity

2018 R&D 100 WINNER

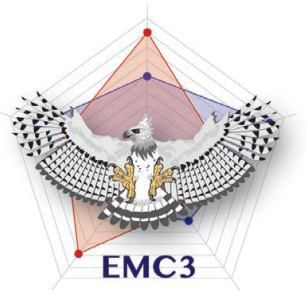
Query

	Find all files in scratch1 as uid 67890		Find all files in lustre scratch1		Find all files in scratch1 and NFS home as uid 67890	
	POSIX	GUFI	POSIX	GUFI	POSIX	GUFI
Files	22,771,329	22,509,652	119,296,067	118,509,899	-	22,522,140
Dirs	240,736	237,759	5,541,230	5,523,153	-	239,603
Time (s)	531.6	14.5	11,309	134.2	-	**14.9**
Files/s	42,835	1,553,956	10,548	883,413	-	**1,511,553**
		36x		**84x**		

Index load

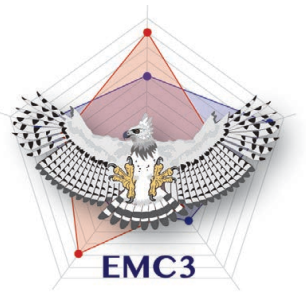
- 118,509,899 files from lustre filesystem into GUFI tree: 148.9s
  - 795,902 files/s
- 13,229,405 files from NFS home filesystem into GUFI tree: 38.4s
  - 344,515 files/s

Why does LANL care about UCX?



- **LANL is actively engaged in developing "in-network computing".**
- **All of the previously mentioned apps can take advantage of UCX**
- **UCX offers a continuum of deployment spaces.**
  - NIC/HCA
  - Phones/IoT
  - Standard applications
    - OpenSHMEM, MPI, PGAS
  - KVS
  - I/O applications
- **UCX offers portability across a very diverse portfolio of systems**
  - We have lots
- **UCX is easily adapted to new and development HW**
  - We are developing new systems and applications/analytics
- **We are currently porting some of the LANL mini-apps to UCX**
- **LANL and key USG partners are supporting UCX for external developers**
- **LANL is working with O&G customers wrt UCX (See EMC^3)**
- **LANL has on-going development projects with Mellanox**
- **LANL is fully supportive of an Open SmartNIC API (OpenSNAPI)**
  - Arm, Mellanox, Broadcom, others?

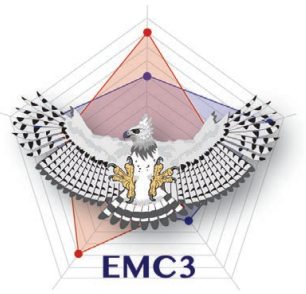
EMC^3 Areas of Interest (where are we doing these things?)



- **Networking**
  - Next Gen Network Requirements/Design
  - Apps/computing/programmability in the network
- I/O & Storage futures
  - Data protection at extremes
  - IO Forwarding
  - Data motion innovation
- Resilient Computing
  - Characterization/prediction
- **High Performance Data Analytics Systems**
  - True performance benchmarks/measurement of HPDA systems
- Inexact computing
  - Characterization for exploration

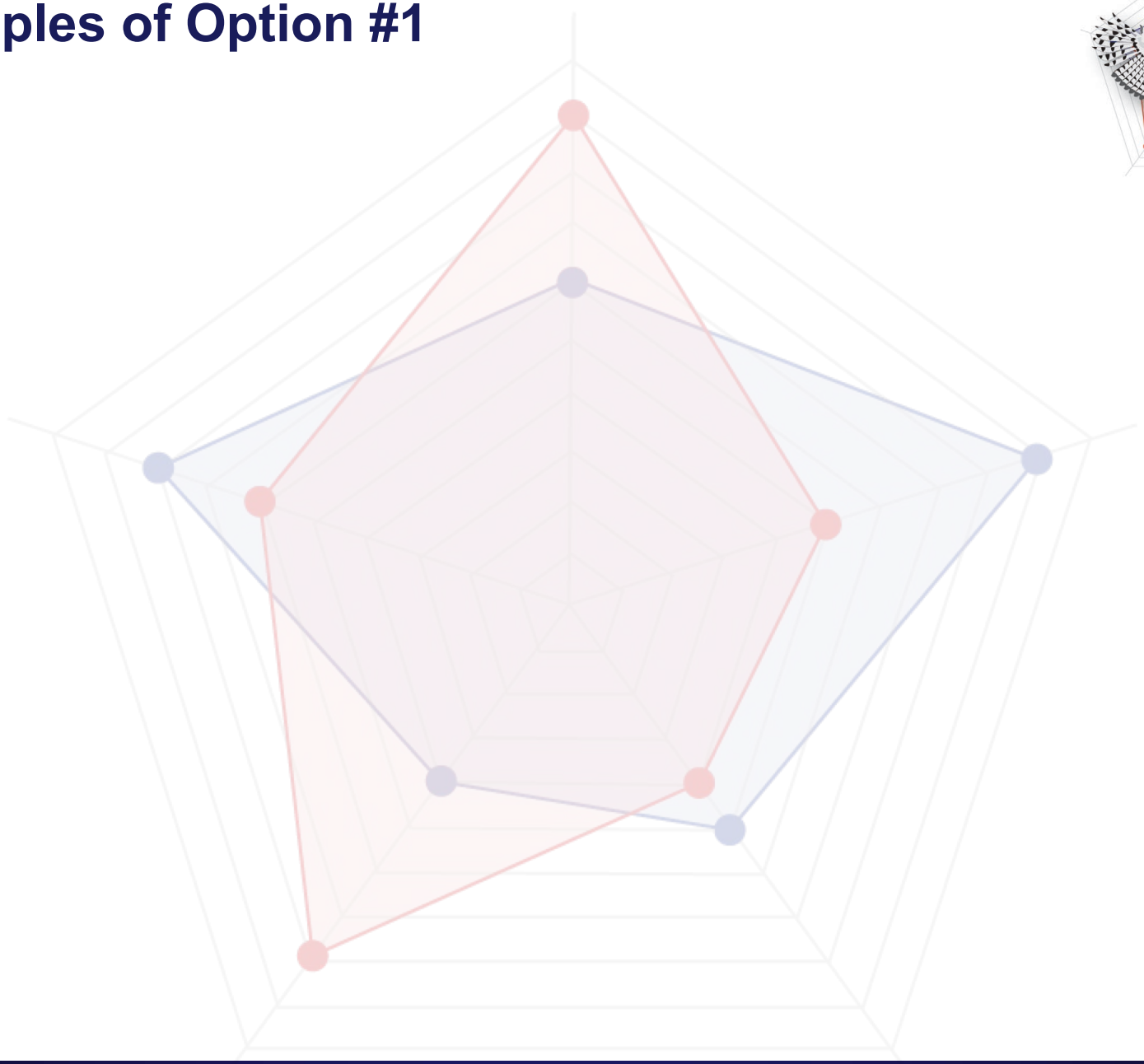
- System SW
  - Next Gen Systems mgmt. (boot/launch/manage/etc.)
  - Leveraging container tech
- HPC Environments
  - Launch, Run Time, Monitoring, Tools innovation
- **Application of ML/DL/AI to HPC**
- **Processor/Memory Complex**
  - Balanced Application focused
  - Power requirements
  - Balanced Performance
  - Scaling
- **Benchmarking and Simulation of Systems**
  - Use of Benchmarks/Simulation against codes to achieve balanced forward progress

OpenSNAPI (potential options/ideas)

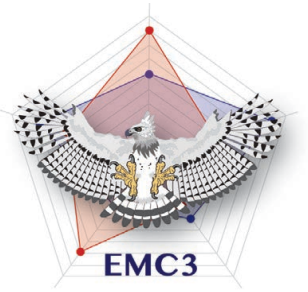


- **Option 1 – Compile and deploy application/parts on SmartNIC**
  - Already doable. (Summer students)
  - Assumes the NIC has an OS, memory, processor. (Just another node)
- **Option 2 – Direct calls into "resident" library**
  - Communication via PCIe/Network interface
  - UCX under OpenSHMEM, OpenFAM
  - CUDA like?
- **Option 3 – Direct "put" of function calls into NIC Space**
  - FPGA like.
    - #PRAGMA and function calls (Xilinx)
  - Assumes driver can set up drop/return spaces.
  - IXP2800 like
  - PCIe or Network? Both?
- **Option 4 – New language?**
  - P4
    - http://lccn.cs.technion.ac.il/wp-content/uploads/2019/02/HeavyHitter-Detection-with-P4-ver-1.2.pdf
  - Micro-C
  - Define one of our own? (LONG PROCESS)
- **Which one do we start with first?**
- **Do we need to assume the NIC is "host" resident? (Sever or serverless)**
  - Yes
    - PCIe or Network comms
  - No
    - Network comms
- **What about programs/functions to the Switch**

Examples of Option #1



TEST PROGRAM #1




```
#include <stdlib.h>
#include "getcpuid.h"

#define DEBUG 1

#define HOST_PROCS 20
#define NIC_PROCS 32
#define HOST_START 0
#define HOST_END (HOST_PROCS - 1)
#define NIC_START (HOST_END + 1)
#define NIC_END (NIC_START + NIC_PROCS)

long psync[SHMEM_REDUCE_SYNC_SIZE];

int main(int argc, char** argv)
{
    shmem_init();
    int np = shmem_n_pes();
    int id = shmem_my_pe();
    int *host_bc = (int*) shmem_malloc(sizeof(int));
    int *nic_bc = (int*) shmem_malloc(sizeof(int));
    char host_name[100];
    gethostname(host_name, 100);
    int cpu_id = get_cpu_id();

    #ifdef DEBUG
    int i, j;
    if(host_bc == NULL || nic_bc == NULL)
    {
        printf("Symmetric pointer == NULL\n\n");
        exit(1);
    }

    for(i = 0; i < np; i++)
    {
        if(shmem_pe_accessible(i) != 1)
            printf("PE %d is unable to access PE %d\n", id, i);
        if(shmem_addr_accessible(nic_bc, i) != 1)
            printf("PE %d is unable to access nic_complete PE %d\n", id, i);
        if(shmem_addr_accessible(host_bc, i) != 1)
            printf("PE %d is unable to access host_complete PE %d\n", id, i);
    }
    shmem_barrier_all();
    #endif

    printf("Hello #1 from process %03d out of %03d, hostname %s, cpu_id %d, host_bc %d, nic_bc %d\n",
        id, np, host_name, cpu_id, *host_bc, *nic_bc);

    shmem_barrier_all();

    if(id == HOST_START)
    {
        *host_bc = 1;
    }
    else if(id == NIC_START)
    {
        *nic_bc = 2;
    }

    shmem_barrier_all();

    shmem_broadcast32(host_bc, host_bc, 1, HOST_START, HOST_START, 0, np, psync);
    shmem_broadcast32(nic_bc, nic_bc, 1, NIC_START, HOST_START, 0, np, psync);

    printf("PE %d completed broadcasts!\n", id);

    shmem_barrier_all();

    printf("Hello #2 from process %03d out of %03d, hostname %s, cpu_id %d, host_bc %d, nic_bc %d\n",
        id, np, host_name, cpu_id, *host_bc, *nic_bc);

    shmem_free(host_bc);
    shmem_free(nic_bc);
    shmem_finalize();

    return 0;
}
```

The Jupiter nodes have 10 cores each. Each core has 2 threads, resulting in a total of 20 Pes per node. Each Jupiter node also has a Bluefield card, which has 16 cores. For this test program we used 10 cores on Jupiter nodes 8 and 9, and all 16 in each of the Bluefield cards, for a total of 52 PEs.

The goal of the program was to explore communication amongst all PEs, understand the relationship between the host nodes and Bluefield cards, and figure out the mapping of PE ranks to physical resources.

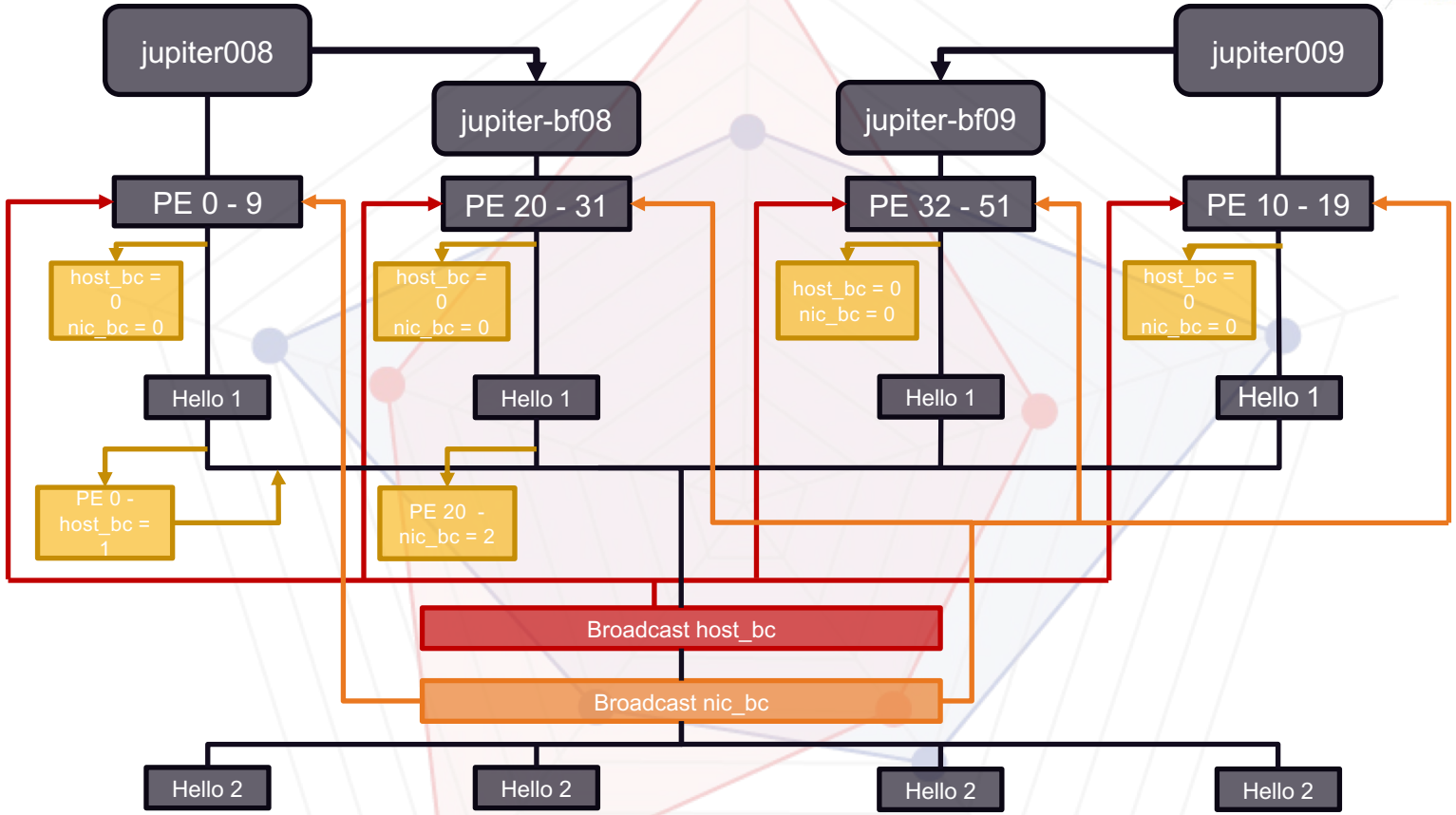
Note that in order to run the program in parallel, the code must be on both the host and BF the cards. The program was run using an appfile, which specified how many cores each node and BF card should use. It was noted that the order of the entries in the appfile, determines the PE ranks.

In the program itself, two symmetric objects were declared: host_bc, and nic_bc, both initialized to 0. These variables were utilized to check for proper synchronization and communication between PE's. A message was printed from each PE showing the initial variable values. The variable host_bc was set to 1 by PE 0 (the first host PE), and the variable nic_bc was set to 2 by PE 20 (the first BF PE).

A broadcast was then sent to update the values of the two variables on all the PEs. Finally, once the broadcasts were complete another message was printed to verify that communication was successful.

Note that the print statements are buffered by the filesystems. However, the broadcast values in the output prove that in reality the four nodes are working in parallel. Therefore, it was concluded that all PEs remained synchronized throughout the process.

The following graphic demonstrates the program's flow with regard to individual processes.


jupiter008
jupiter-bf08
jupiter-bf09
jupiter009
PE 0 - 9
PE 20 - 31
PE 32 - 51
PE 10 - 19
host_bc = 0 nic_bc = 0
host_bc = 0 nic_bc = 0
host_bc = 0 nic_bc = 0
host_bc = 0 nic_bc = 0
Hello 1
Hello 1
Hello 1
Hello 1
PE 0 - host_bc = 1
PE 20 - nic_bc = 2
Broadcast host_bc
Broadcast nic_bc
Hello 2
Hello 2
Hello 2
Hello 2

Test Program #2

```c
#include <stdio.h>
#include <unistd.h>
#include <shmem.h>

int main(void)
{
        int me, npes;
        shmem_init();
        int *sum = (int*) shmem_malloc(sizeof(int));
        me = shmem_my_pe();
        npes = shmem_n_pes();
        char hostname[100];
        gethostname(hostname, 100);


        if (me > 0) {
                shmem_int_atomic_add(sum, me, 0);
        }


        shmem_barrier_all();

        if (me == 0) {
                printf("From hostname %s: Sum from 1 to %d = %d\n", hostname, npes - 1, *sum);
        }

        shmem_finalize();

        return 0;
}
"sum2n.c" 82L, 2655C
```

For the second test program we used 20 cores on both Jupiter nodes 8 and 9, and all 16 in each of the Bluefield cards, for a total of 72 PEs.

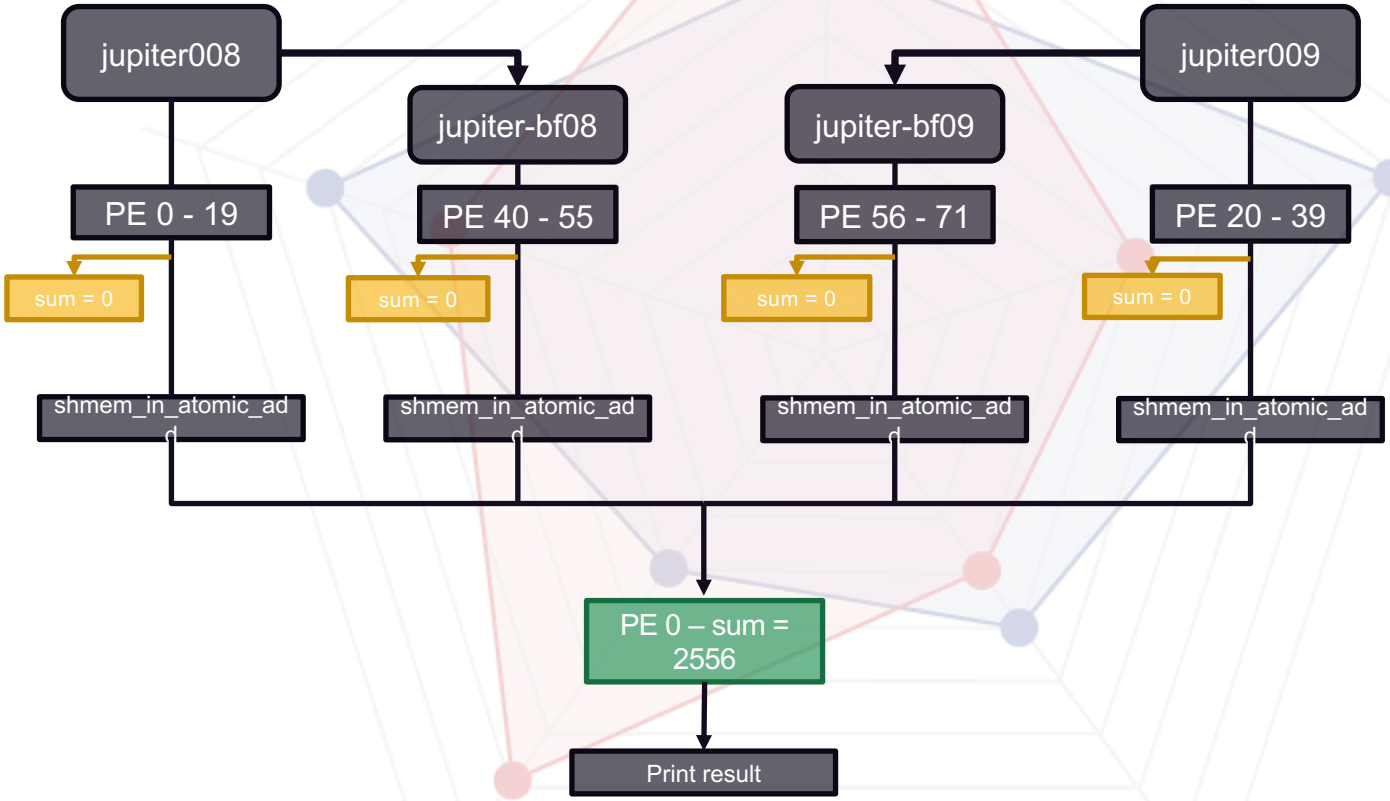
The goal of this simple program was to test the use of atomic operations across PEs on the Jupiter hosts and the Bluefield cards.

Just as the previous test, this program was run using an appfile.

This program atomically adds all PE rank values and stores the result into the symmetric object “sum” on PE 0. Once all rank values have been added, the result is printed.

Based on the output, it was concluded that atomic operations are correctly executed across all PEs.

The following graphic demonstrates the program’s flow with regard to individual processes.


jupiter008
jupiter-bf08
jupiter-bf09
jupiter009
PE 0 - 19
PE 40 - 55
PE 56 - 71
PE 20 - 39
sum = 0
sum = 0
sum = 0
sum = 0
shmem_in_atomic_add
shmem_in_atomic_add
shmem_in_atomic_add
shmem_in_atomic_add
PE 0 – sum = 2556
Print result

Exploring Mellanox Bluefield SmartNICs as Accelerators for Heterogeneous Architectures

Brody Williams*, Liliana Aguirre-Esparza†, Wendy Poole‡, Stephen Poole‡
*Texas Tech University †New Mexico State University ‡Los Alamos National Laboratory

Motivation

Limited advances in processor technology in recent years have forced researchers to explore alternative techniques to provide continued system performance improvements and facilitate further scaling. Many resulting approaches have shifted away from the strictly CPU-centric approach used in the past in favor of more heterogeneous architectures. These architectures often employ added computational components to support offloading computation from the CPU. Frequently, these components are also paired with their own local memory in order to minimize performance degradations associated with data movement.

In this work, we propose an extension to the heterogenous architecture paradigm using Mellanox Bluefield SmartNICs. These devices combine state of the art network controllers together with 16 ARM cores into a device that provides unique potential. Herein, we explore the feasibility of utilizing these SmartNICs as accelerators capable of offloading both communication routines, as well as

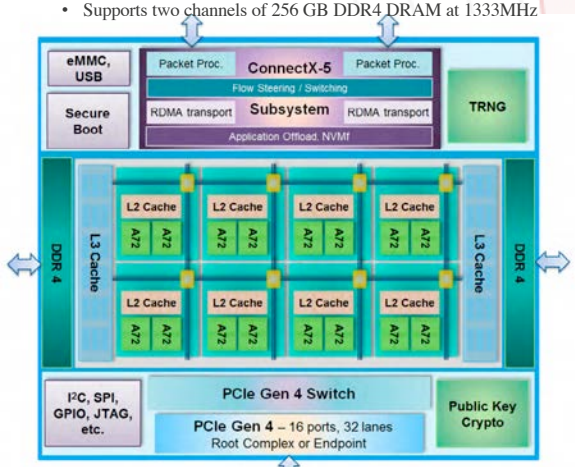
Bluefield System on Chip Architecture

16 ARMv8 Cortex-A72 Cores
- Three-level cache hierarchy
- SkyMesh coherent interconnect
- 128b ARM Neon SIMD execution unit per core

Connect X-5 Subsystem
- Dual Virtual Protocol Interconnect (VPI) ports
- Ethernet/Infiniband at 100Gbps per port
- RDMA & NVMe-oF support

Integrated PCIe Switch
- 32 bifurcated PCI 4.0 lanes
  - Configurable as 2x16/4x8/8x4/16x2
  - Speeds up to 200Gbps

Memory Controllers
- Supports two channels of 256 GB DDR4 DRAM at 1333MHz



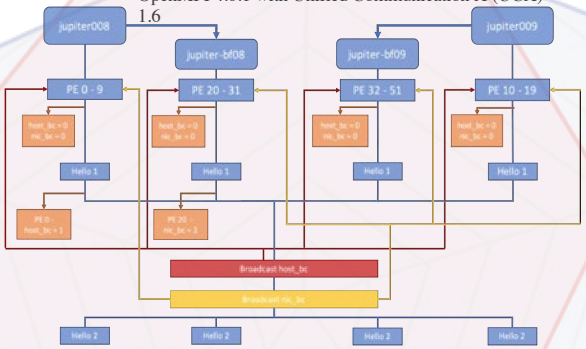
Bluefield System on Chip Architecture.
http://www.mellanox.com/related-docs/prod_multi-core_processors/PB_Bluefield_SoC.pdf

Communication Experiment

In order for the SmartNICs to perform effectively as accelerators at any useful scale, they need to be able to communicate with other devices. Therefore, as a necessary prerequisite to any application performance optimization attempt, we first conducted an experiment to determine compatibility between the ARM cores onboard each SmartNIC and prominent distributed-address space programming paradigms.

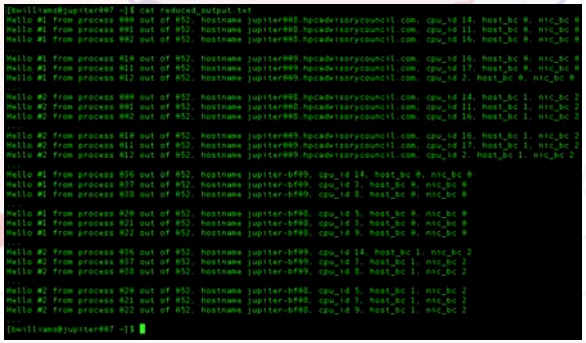
Mellanox Testbed – jupiter007 - jupiter010

- Intel Xeon E5-2680 v2 10-core processors
- 64 GB of memory
- Paired Bluefield SmartNIC with 16GB onboard memory
- CentOS 7
- OpenMPI 4.0.1 with Unified Communication X (UCX) 1.6



Experiment Program Flow

Output from our experiment shows that processor cores are properly utilized across multiple nodes and SmartNICs. Further, correct broadcast variable values demonstrate that proper inter-device communication and synchronization takes place despite buffered print statements. (Note that the output has been simplified for presentation purposes.)



Output Screenshot

Acceleration Opportunities

Asynchronous Execution of Task-Parallel Code Segments
- Map host and SmartNIC cores to orthogonal tasks

EMC3

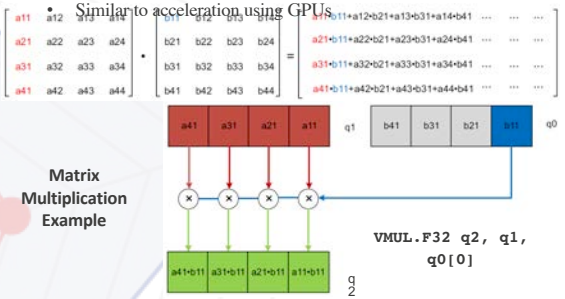
Algorithm 1: Sample Execution on Host

```
for i ← 0 to N do
    Host_Proc(Compute(i))
    Host_Proc(Verify(i))
end
```

Algorithm 2: Sample Execution on Host + SmartNIC

```
for i ← 0 to N + 1 do
    if Host_Proc then
        if i ≠ (N + 1) then
            Host_Proc(Compute(i))
    else if SmartNIC_Proc then
        if i ≠ 0 then
            SmartNIC_Proc(Verify(i − 1))
end
```

Vectorization of SIMD Operations using Neon Units
- Individual Neon unit per ARM core prevents resource contention
- Similar to acceleration using GPUs



http://infocenter.arm.com/help/topic/com.arm.doc.dui0489f/DUI0489F_arm_assembler_reference.pdf

Inter-Process Communication Calls
- Offload routines to SmartNIC cores
- Prevent blocking of host CPU cores
- Perform buffering and collective computation locally on SmartNICs

Future Work

Our preliminary work for this project indicates that acceleration using Mellanox Bluefield SmartNICs is feasible. Further, our communication experiment demonstrates that the SmartNICs are fully compatible with the already widely adopted OpenSHMEM and MPI standards. Our future work will focus on determining whether or not these SmartNICs perform effectively as accelerators, and, if so, how best to optimize and deploy code for SmartNIC acceleration.

In particular, we plan to first optimize several provided Department of Defense benchmarks using the methods discussed above. We are also interested in investigating the use of active messages to pass fully dependence-free code segments to the SmartNICs for execution. Finally, when more familiar with writing code for SmartNIC acceleration, we plan to explore developing a library that abstracts SmarNIC acceleration away from the software developer while providing optimal performance.

Acknowledgements

We would like to thank the Department of Defense for supporting this project. We are also grateful to Mellanox Technologies, Parks Fields, and John Snyder of Duke University for their collaborative efforts.

Investigating the use of BlueField with OpenSHMEM and MPI over OpenUCX distributed applications

Liliana Aguirre Esparza, New Mexico State University
lili91@nmsu.edu

Mentor: Wendy K. Poole, HPC-ENV
wkpoole@lanl.gov

Co-mentor: Steven Poole, ALDSC/HPC
swpoole@lanl.gov

Introduction

As we move into the future computer systems and data center continue to grow and evolve. With this growth comes the issue of performance. Today's data center servers must deal with very large storage and computer workloads, which spend very valuable CPU cores on background processing, instead of application computations and processing. This in turn results in a decrease in performance. For this reason, it is crucial that we address these issues and find new ways to relieve the pressure off CPU cores that should be used for program processing.

Goal

The objective of this summer's research was to begin the development of a methodology that will allow us to profile, gather and analyze any prospective computations from kernel from both the Department of Defense (DoD) and LANL, that may be offloaded to the BlueField Multicore System of Chip to improve performance. The goal was to build and run the provided benchmark, which is written with both MPI and OpenSHMEM as communication libraries and gain an understanding of the application and then potentially take the initial steps towards creating a library for Smart NICs.

The BlueField Multicore System On A Chip (SoC) and Controller Card [1]

BlueField SoC Features

Bluefield is a highly integrated system on chip (SoC), which is optimized for NVMe storage systems, Network Functions Virtualization (NFV), security systems, and embedded applications. The BlueField SoC was developed by Mellanox Technologies as a way to address performance, network and cybersecurity concerns. The chip includes:

- a set of 64-bit Armv8 A72 core CPUs,
- a PCIe switch,
- and a network controller.

The BlueField SoC was developed as a solution for building Just-A-Bunch-Of-Flash (JBOF) systems, All-Flash-Array and storage applications for NVMe over Fabrics. The PCIe switch on the BlueField supports up to 32 lanes of the Gen3 and 20 Gb/s transfers of the Gen4, which allows

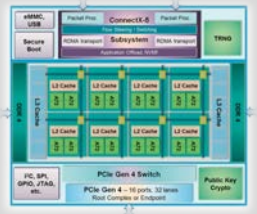


Figure 1. BlueField Architecture

transfers of 16 Gb/sec of data to and from the SSDs.

Controller Card

The Mellanox BlueField Controller Card is designed to control and take advantage of all the features of the BlueField SoC. The card has a PCIe standard form factor, and can be used as a high performance networking card.

Figure 2. BlueField SoC

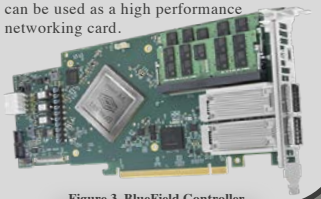


Figure 3. BlueField Controller Card

Benchmarking

Several profiling tests were performed on a DoD application using a sampling tool, in order to understand which part of the application was allocated the most time. The application was written to support both MPI and OpenSHMEM, therefore each test was duplicated to account for both platforms. The tests were performed on both Trinitite and Capulin in order to observe the effect of memory and number of cores on the application performance.

Methodology

The first set of tests were performed on Trinitite and Capulin using 2 nodes. Each node on Trinitite has 128GB of memory and 32 Haswell cores, and each node on Capulin has 256GB of memory and 56 cores. The tests were done using 25%, 50%, and 75% memory. For each memory percentage three different tests were implemented using diverse program arguments. A second set of tests was performed using 4 nodes, and a third using 8 nodes.

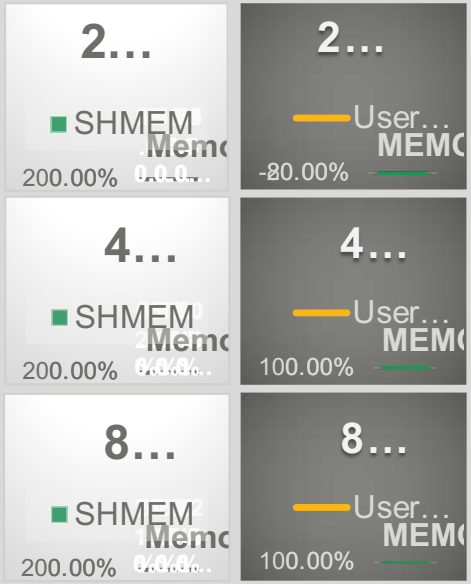


Figure 4. Capulin Test 1 Results

Profiling Results

The tests on Trinitite revealed no clear pattern. The results varied based on the memory percentage and the program arguments. However, the results on Capulin yielded a clearer pattern. Based on these results it was concluded that time spent on communication between PEs using both OpenSHMEM and MPI increased in correlation to the number of cores. The profiling results for test one with OpenSHMEM at 25%, 50%, and 75% memory, using 2, 4 and 8 Capulin nodes are displayed in Figure 4. A similar pattern was shown for tests 2 and 3.

BlueField Testing

A simple program was used to test the use of atomic operations across PEs on the host nodes and the BF cards. The code was adapted from a program found on the official OpenSHMEM website.[2]

Testing Environment

The system used to perform these tests was made up of several types of nodes. For our purposes we used the Jupiter nodes with BlueField cards attached. The Jupiter nodes consist of 10 cores, each with 2 threads, resulting on a total of 20 PEs per node. Each Jupiter node also has one BlueField controller card and SoC, which consists of 16 cores.

Test Program

For the test program, shown in Figure 5 we used 20 cores on both Jupiter nodes 8 and 9, and all 16 in each of the Bluefield cards, for a total of 72 PEs. It was noted that in order to run the program in parallel, the code must be on both the host and BF the cards. The program was ran using an appfile specifying how many cores each node and BF card should use to run the program. The test program atomically adds all PE rank values and stores the result into the symmetric object "sum" on PE 0. Once all rank values have been added, the result is printed.

Figure 5. Test Program Code

Results and Observations

After the running the program it was noted that the order of the entries in the appfile, determines the PE ranks. Based on the output, it was concluded that atomic operations are correctly executed across all PEs. Figure 6 demonstrates the program's flow with regard to individual processes.

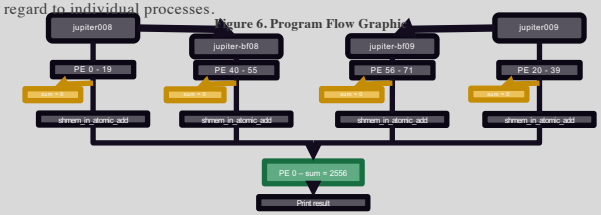


Figure 6. Program Flow Graphic

Future Work

The profiling data gathered gives us a greater understanding of the provided application and allows us to hypothesize about what computations may be offloaded to improve performance. Based on the results, a good place to start offloading is communication computations. Furthermore, the information gathered about how the BlueField relates to the host node will allow us to find the best way to break down the application to so the BlueField can handle specific computation in a way that actually improves performance. The next step would be to perform the same profiling tests from Trinitite and Capulin on a system equipped with BF cards and compare those results to the previously gathered

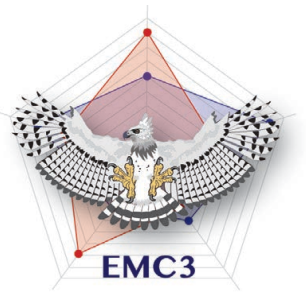
Acknowledgements

I would like to acknowledge the mentoring I have received from Wendy Poole, Steven Poole, and Brody Williams. I also would like to acknowledge the Department of Defense as well as Mellanox Technologies for the support to this project.

References

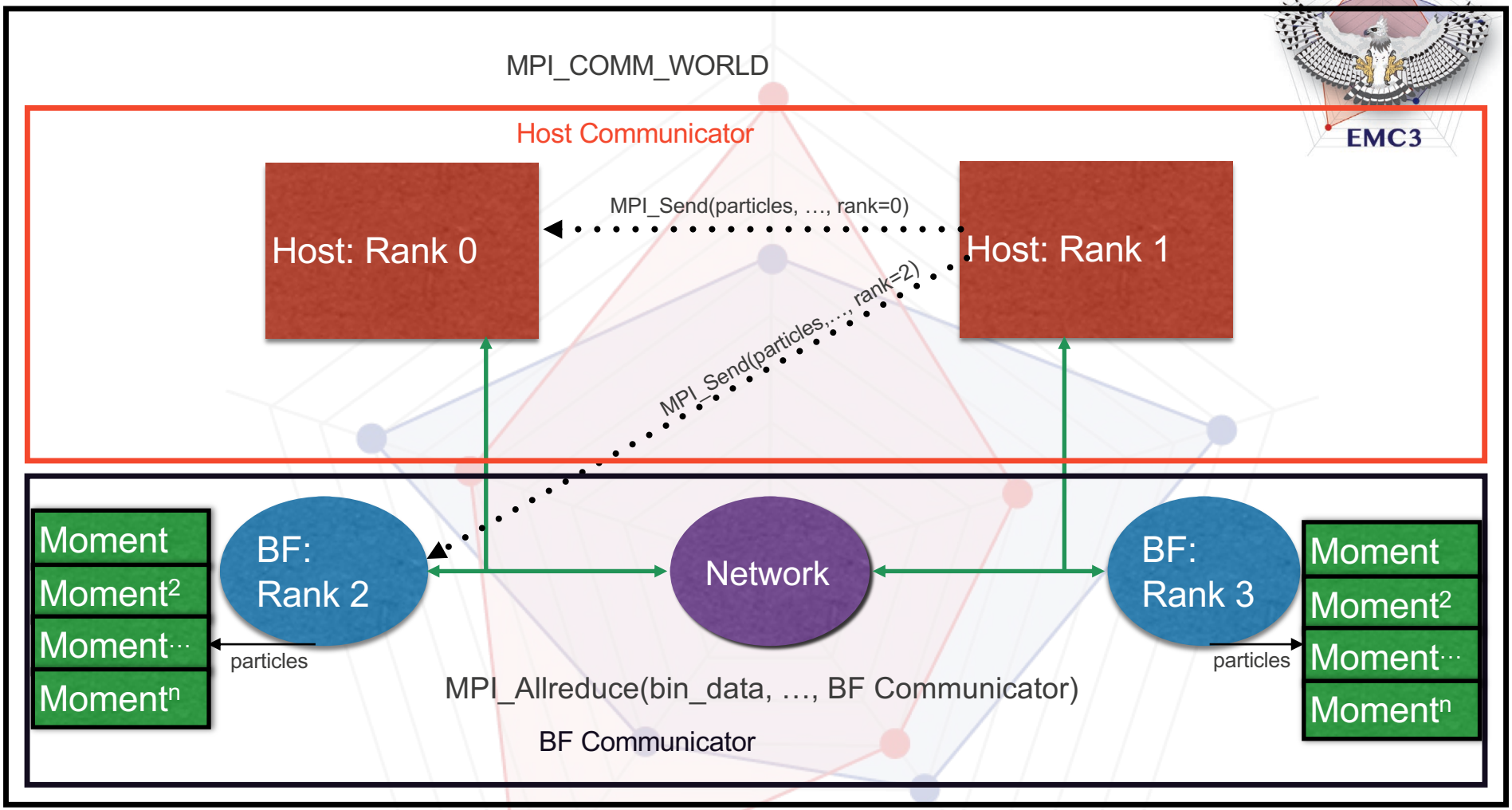
[1] Mellanox Technologies, http://www.mellanox.com/products/bluefield-overview/
[2] OpenSHMEM.org, https://github.com/openshmem-org/openshmem-examples/blob/master/C/sum2a.c

VPIC Problem (Courtesy Brad Settlemyer)



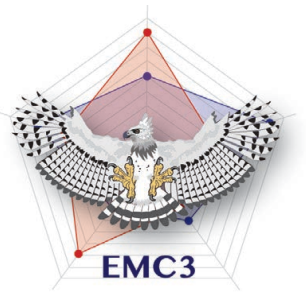
- Problem Statement
  - Consider the problem of tracking a spatially correlated energy peak through space, i.e. tracking an energy wave. In a particle-in-cell code, such as LANL's VPIC, the energy values are calculated using trillions of small particles (approximately 32 Bytes each) that move between processes as the simulation progresses. In order to identify all energy peaks at a timestep it is necessary to understand the energy distribution of the particles. The approach most commonly used in practice is to post-process the output data sets and construct energy histograms. The primary challenge in constructing an energy histogram is that the histogram bin widths are typically impossible to estimate in advance. To determine appropriate bin widths during post-processing requires using computational resources similar to the original simulation in order to efficiently post-process all of the simulation output. Another alternative is to simply sample some small percentage of the particles and estimating an empirical distribution based on the sampled particles. This technique is problematic because simulations are typically constrained by memory use, and both the all-to-all communications and buffering required to sample the energy space are expensive to achieve in practice.

Integrating Bluefields into VPIC (Jack Snyder – Mellanox/DUKE/LANL)



- BF's are assigned an MPI Rank like any other host
- Particles are exchanged with MPI when they cross boundaries
- BF is also sent particles, which are then binned
- BF calculates Moments for each particle
- BF reconstructs quantiles of particle distribution from Moments

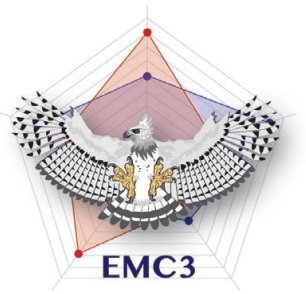
Option #2 (Some potential instructions)




```
SNICresult status = snicInit(0);
status = snicDeviceGet(&dev, 0);
status = snicDeviceComputeCapability(&major,&minor,dev);
snicGetDeviceCount(&device_count);
snicSetDevice( device );
snicGetDevice(&device);
snicGetDeviceProperties(&deviceProp,device);
snicDeviceReset();
snicDriverGetVersion ( int* driverVersion )
snicDeviceGetAttribute ( int* pi, snicDEVICE attrib, snicDEVICE_dev )
snicDeviceGetName ( char* name, int  len, snicDEVICE dev )
snicDeviceTotalMem ( size_t* bytes, snicDEVICE dev )
```

NVIDIA has MANY great potential examples for this mode.
I copied these from NVIDIA docs.

Option #3 (Some potential examples)



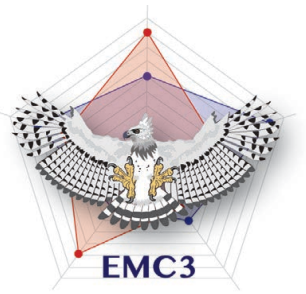
- http://xillybus.com/tutorials/vivado-hls-c-fpga-howto-2
- Can use #pragma

```
void snicBUS_wrapper(int *in, int *out)
{
#pragma AP interface ap_fifo port=in
#pragma AP interface ap_fifo port=out
#pragma AP interface ap_ctrl_none port=return
    snic_puts("Hello, world\n"); // Handle input data
}
```

- Can write directly to device
  - fdr = open("/dev/snicBUS_read_32", O_RDONLY);
  - fdw = open("/dev/snicBUS_write_32", O_WRONLY);
  - write(fdw, (void *) &tologic, sizeof(tologic));
  - read(fdr, (void *) &fromlogic, sizeof(fromlogic));

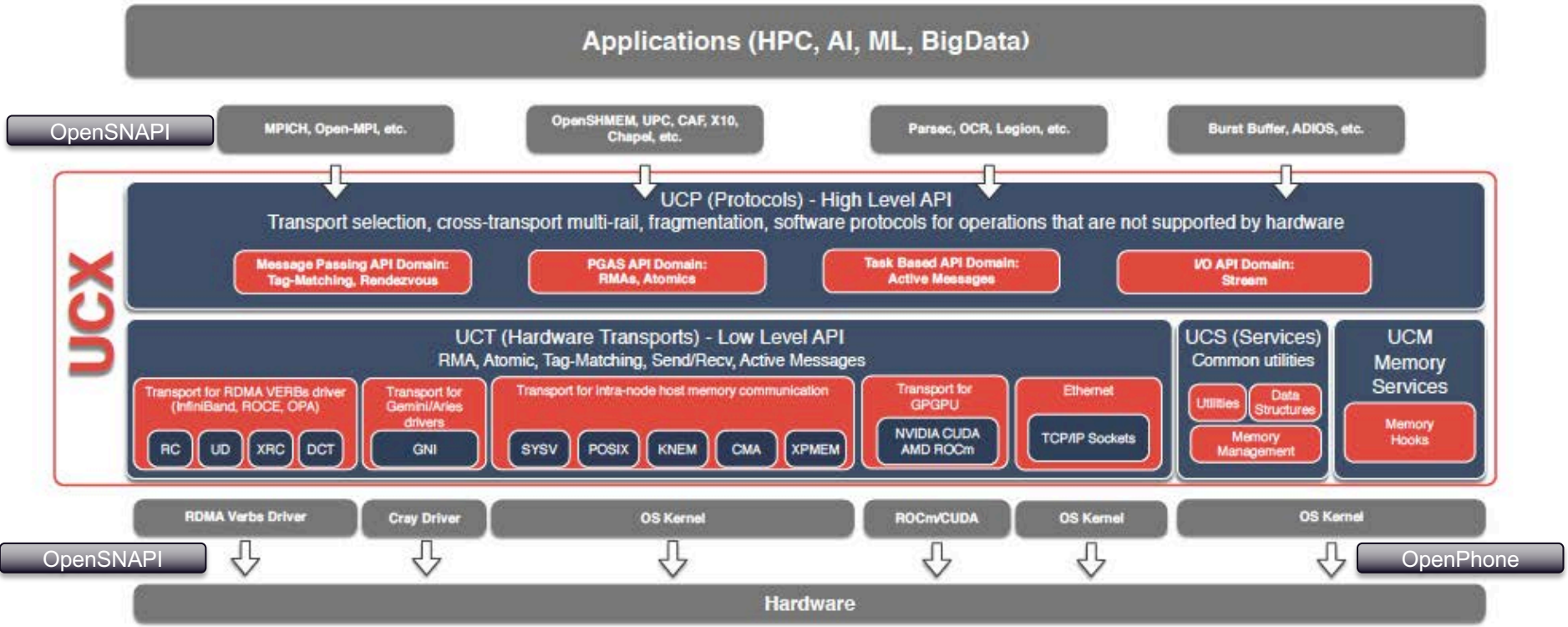
- Mellanox and others have many examples on how to use at this level.

Option #4 (Some potential examples)



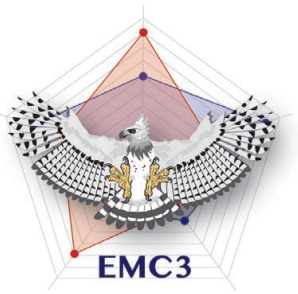
- Do we define our own? (Language)
- Do we adopt something like P4 or Micro-C
- UNO
  - SDN-controlled NF offload architecture (wisc)
  - https://wisr.cs.wisc.edu/papers/p506-le.pdf
- Floem
  - a language, compiler, and runtime — for programming NIC-accelerated appli- cations.
  - https://www.usenix.org/system/files/osdi18-phothilimthana.pdf
- What about Python?
- What can we learn from SDN?
  - Can we adopt/adapt a larger communities work?
- How do we make it fit best with UCX?
- https://blog.mellanox.com/2018/09/why-you-need-smart-nic-use-cases/
- Fund a summer intern(s) to develop summary paper(s) on:
  - Vendors
  - Languages
  - Functionalities
  - Classifications

Potential future UCX Diagram



Where would OpenSNAPI fit in the UCX “eco-system”?

Contributors



- **Bill Archer** — **Physics**
- **Jerry Brock** — **Dep. ASC PM**
- **Jim Cruz** — **Graphics Design**
- **AnnMarie Cutler** — **Graphics**
- **Gary Grider** — **All Things I/O - EMC^3**
- **Paul Henning** — **Algorithms**
- **Beth Kaspar** — **ALDSC PM**
- **Wendy Poole** — **Analytics**
- **John Sarrao** — **Physics**
- **Pavel Shamis** — **Arm**
- **Janet Mercer-Smith** — **R&D100**
- **Kevin Sutton** — **Photographer**
- **Justin Warner** — **R&D100**

- **Liliana A. Esparza** — **LANL Summer Intern (NMSU)**
- **John "Jack" Snyder** — **Mellanox Summer Intern (Duke)**
- **Brody Williams** — **LANL Summer Intern (TxTech)**
- **Richard Graham** — **Mentor**
- **Steve Poole** — **Mentor**
- **Wendy Poole** — **Mentor**

Questions?


CRAY
Alliance for Computing at Extreme Scale
NNSA
Los Alamos
Sandia National Laboratories
TRINITY

National Nuclear Security Administration
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA