OAK RIDGE National Laboratory

Architecting and Programming of Future Extremely Heterogeneous Systems

Jeffrey S. Vetter

With many contributions from FTG Group and Colleagues

MODSIM 2019
Seattle
14 Aug 2019

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Future Technologies Group ornl

http://ft.ornl.gov vetter@computer.org

DARPA Domain-Specific System on a Chip (DSSoC) Program

DARPA ERI DSSoC Program: Dr. Tom Rondeau

OAK RIDGE National Laboratory

Intelligent Scheduling | Domain Representation | MAC | Software

The history of GNU Radio inspiring the key technical areas of DSSoC

2001 Single core, single thread (Pentium 4)


commons.wikimedia.org


www.mccdaq.com

2004 Growing with technology (Pentium D)


commons.wikimedia.org


media.wired.com

2008 Multi-threaded scheduling (Cell Broadband Engine)

commons.wikimedia.org

2008 Moving to multi-core processors (i7)


commons.wikimedia.org

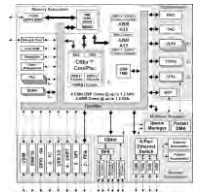

www.ettus.com

2013 Embedded computing (Arm and Xilinx) Introduction of VOLK

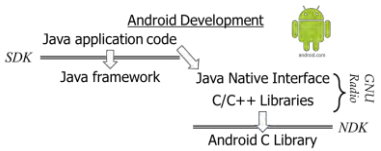

libvolk.org/

2014 Porting to other SoCs (TI Keystone II)

processors.wiki.ti.com


www.ti.com

2015 Porting to the smart phone (Android devices)



Today

Processors	Computer	I/O
Intel *Arm* *PowerPC*	*Server farms* *Workstations* *Laptops* *Smart Phones* *Embedded Systems*	*USB 2.0* *USB 3.0* *Thunderbolt2* *1 GigE* *10 GigE* *PCIe x4*

Slide courtesy of Dr. Tom Rondeau, DARPA MTO

Getting the best out of specialization when we need programmability

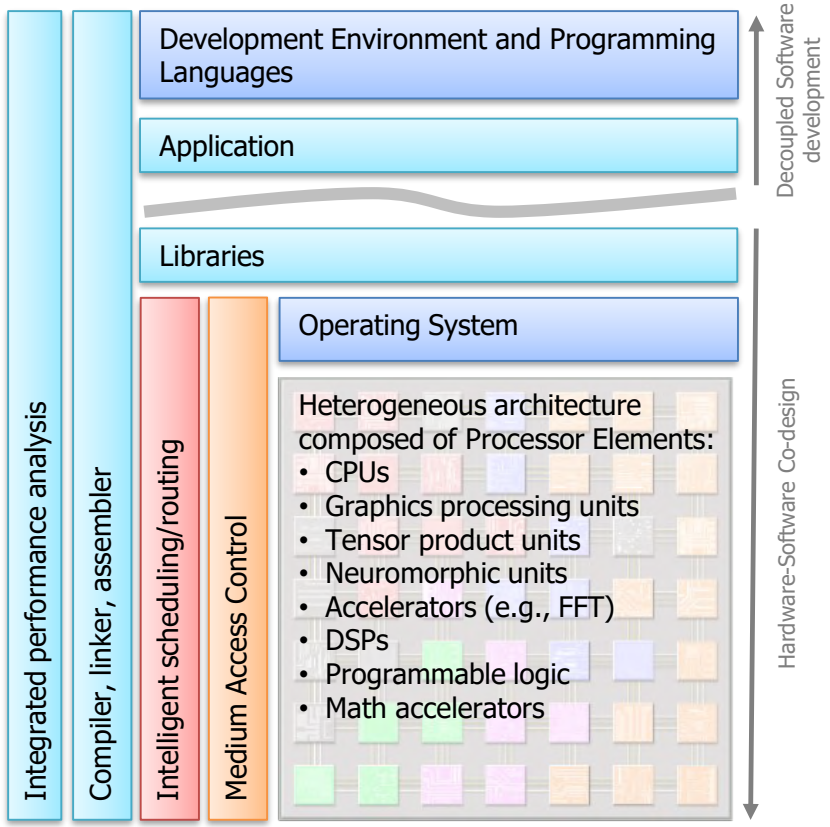
Three Optimization Areas

1. Design time
2. Run time
3. Compile time

Addressed via five program areas

1. Intelligent scheduling
2. Domain representations
3. Software
4. Medium access control (MAC)
5. Hardware integration

DSSoC's Full-Stack Integration



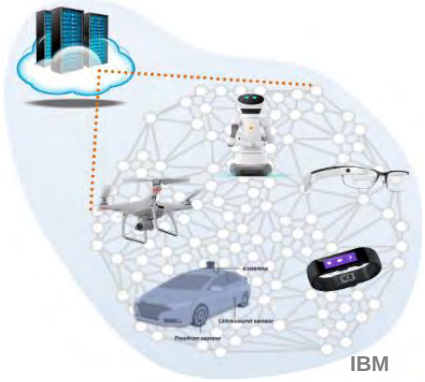
Looking at how Hardware/Software co-design is an enabler for efficient use of processing power

Slide courtesy of Dr. Tom Rondeau, DARPA MTO

DSSoC performer domains and applications

IBM T. J. Watson Research Center
Pradip Bose
Columbia University, Harvard University, Univ. of Illinois at Urbana-Champaign

CV+SDR
- Multi-domain application
- Multi-spectral processing
- Communications



IBM

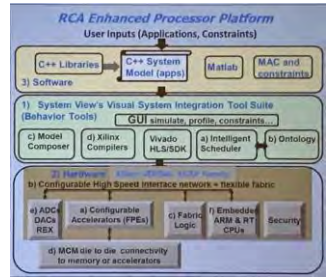
Arizona State University
Daniel W. Bliss
Univ. of Michigan, Carnegie Mellon University, General Dynamic Mission Systems, Arm Ltd., EpiSys Science

SDR
- Unmanned aerial
- Small robotic & leave-behind
- Universal soldier systems
- Multifunction systems

PlastyForma

Raytheon/Xilinx
Tom Kazior

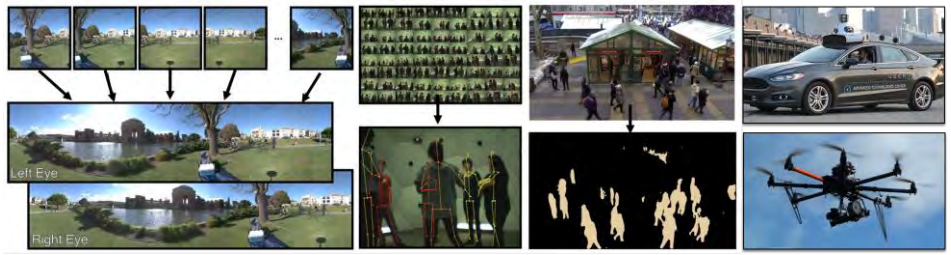
SDR
- Xilinx ACAP
- Visual system integrator
- Improved reconfigurability of processing



Raytheon

Stanford University
Mark Horowitz
Clark Barrett, Kayvon Fatahalian, Pat Hanrahan, Priyanka Raina

Computer Vision
- Still image and video processing
- Autonomous navigation
- Continuous surveillance
- Augmented reality

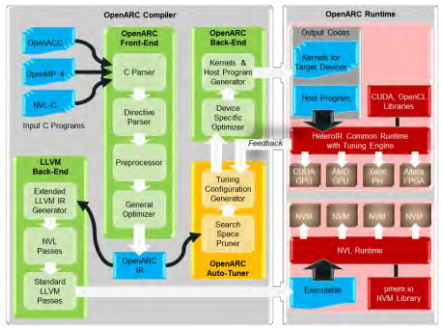


Stanford

Google/YouTube

Oak Ridge National Laboratory
Jeffrey Vetter

SDR
- Communications and signal processing focused
- Up-front processing / data cutdown
- Improving understanding of processing systems



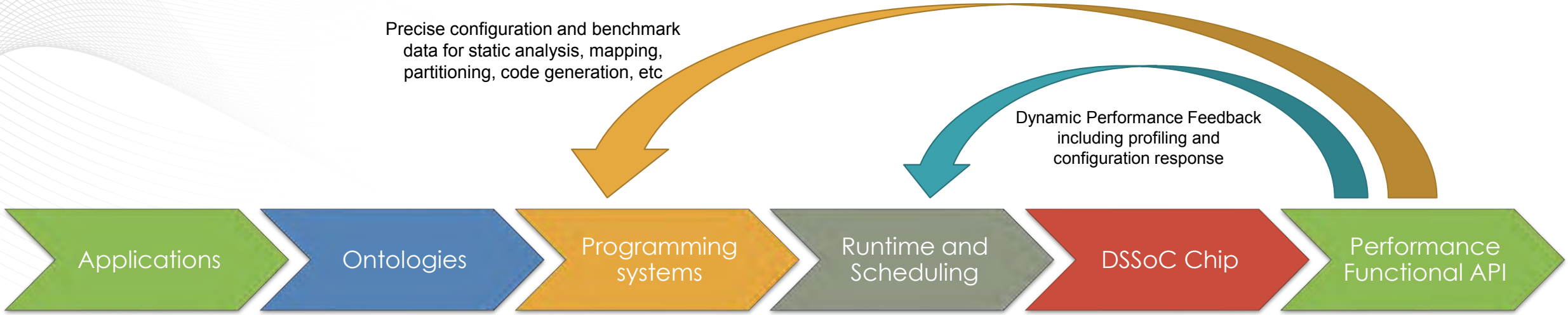
ORNL

Slide courtesy of Dr. Tom Rondeau, DARPA MTO

ORNL Cosmic Project

Jeffrey S. Vetter
Seyong Lee
Mehmet Belviranli
Roberto Gioiosa
Richard Glassbrook
Abdel-Kareem Moadi

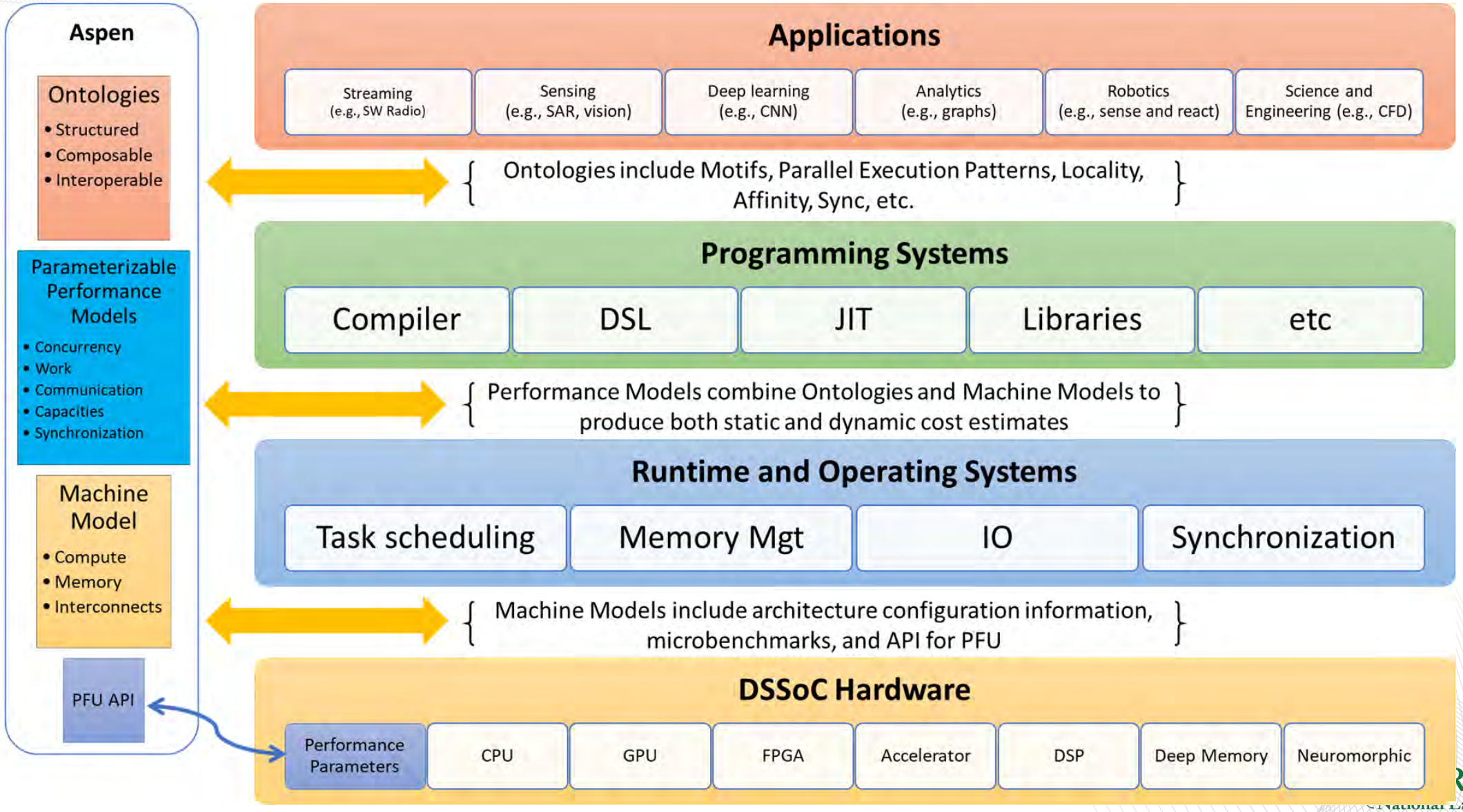
Development Lifecycle



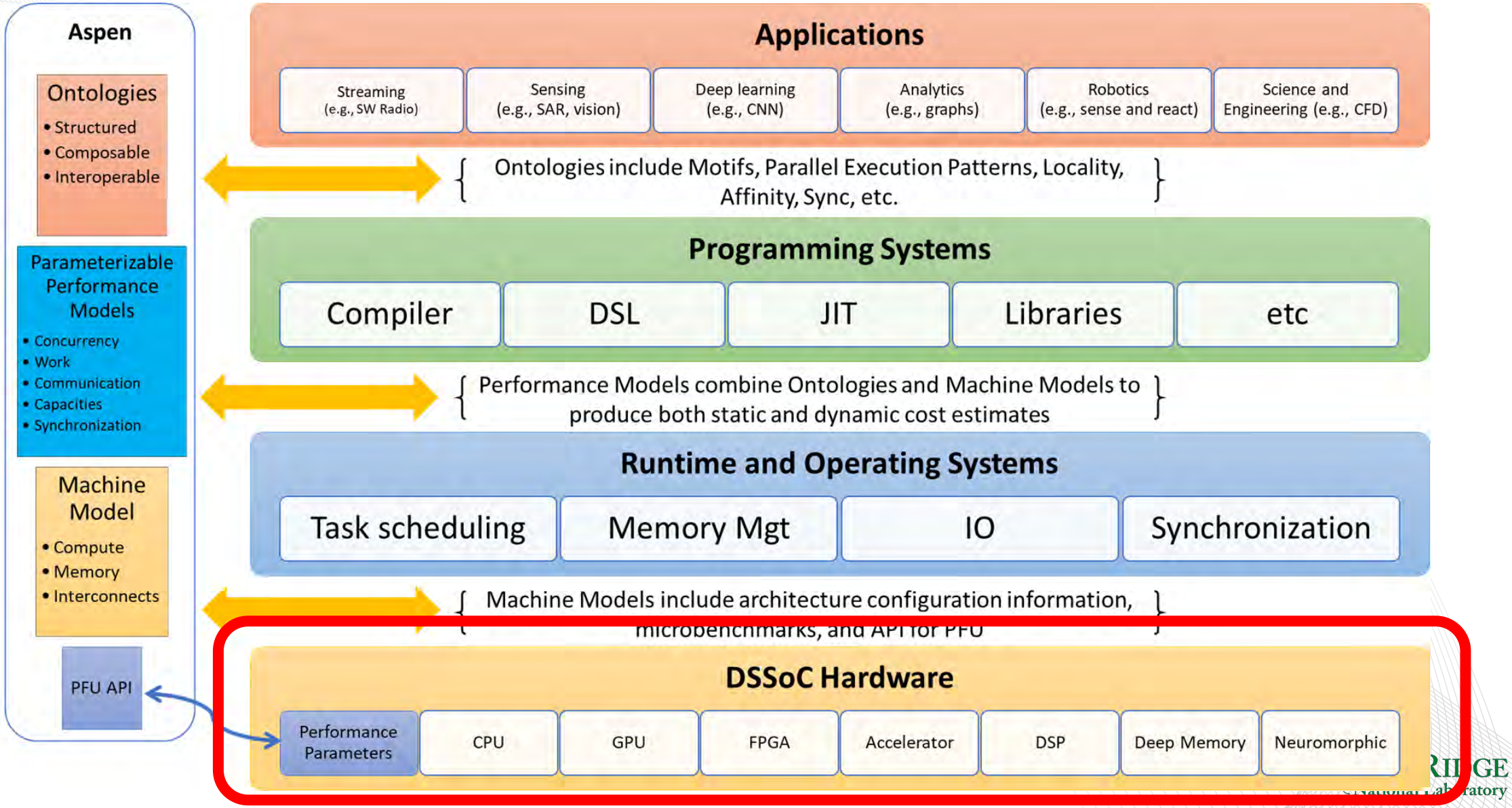
- Create scalable application Aspen models manually, with static or dynamic analysis, or using historical information
- Ontologies based on Aspen models using statistical and machine learning techniques
- Programming systems built to support ontologies
- Query Aspen models and PFU for automatic code generation, optimization, etc.
- Intelligent runtime scheduling uses models and PFU to inform dynamic decisions
- Dynamic resource discovery and monitoring
- DSSoC design quantitatively derived from application Aspen models
- Early design space exploration with Aspen
- As feature of DSSoC, PFU API provides dynamic performance response of deployed DSSoC to intelligent runtime and programming system.

- DSSoC design quantitatively derived from application Aspen models
- Early design space exploration with Aspen

Cosmic Castle | Project Overview



Cosmic Castle | Project Overview

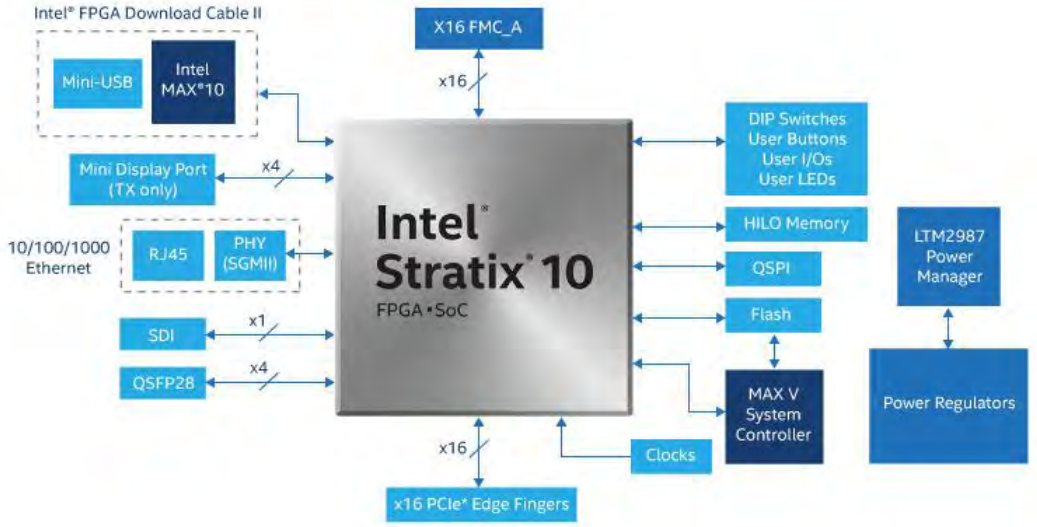


NVIDIA DGX Workstation

- 4X Tesla V100 GPUs
- TFLOPS (Mixed precision) 500
- GPU Memory 128 GB total system
- NVIDIA Tensor Cores 2,560
- NVIDIA CUDA® Cores 20,480
- CPU Intel Xeon E5-2698 v4 2.2 GHz (20-Core)
- System Memory 256 GB RDIMM DDR4
- Full NVIDIA stack
- Other compilers/tools installable on request

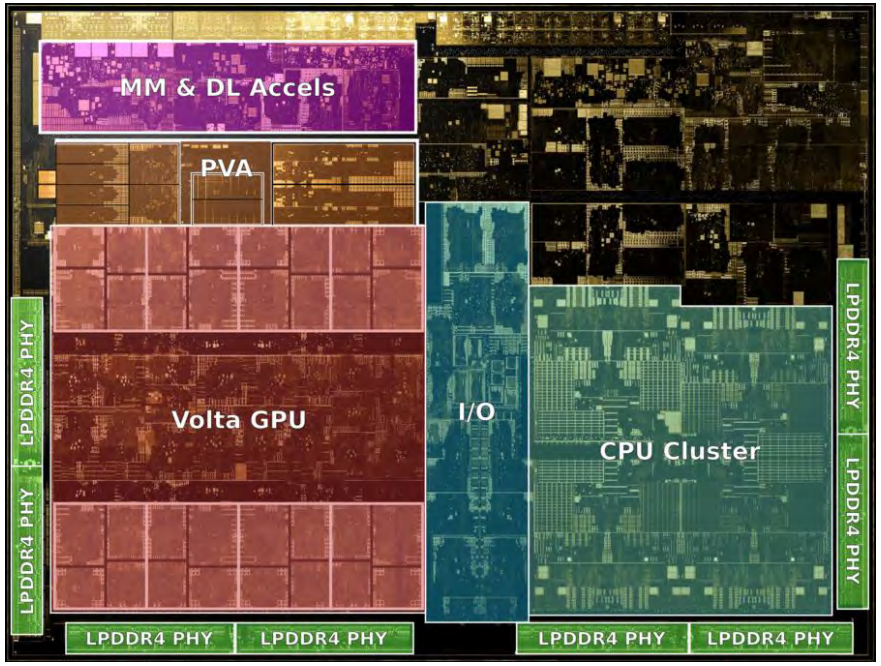
Intel Stratix 10 FPGA

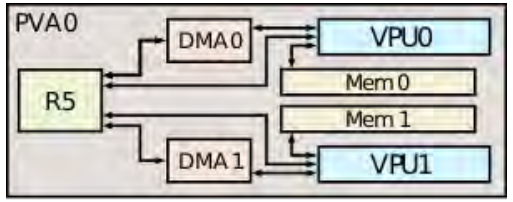
- Intel Stratix 10 FPGA and four banks of DDR4 external memory
  - Board configuration: Nallatech 520 Network Acceleration Card
- Up to 10 TFLOPS of peak single precision performance
- 25MBytes of L1 cache @ up to 94 TBytes/s peak bandwidth
- 2X Core performance gains over Arria® 10
- Quartus and OpenCL software (Intel SDK v18.1) for using FPGA
- Provide researcher access to advanced FPGA/SOC environment

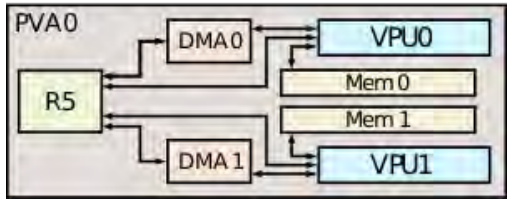


NVIDIA Jetson AGX Xavier SoC

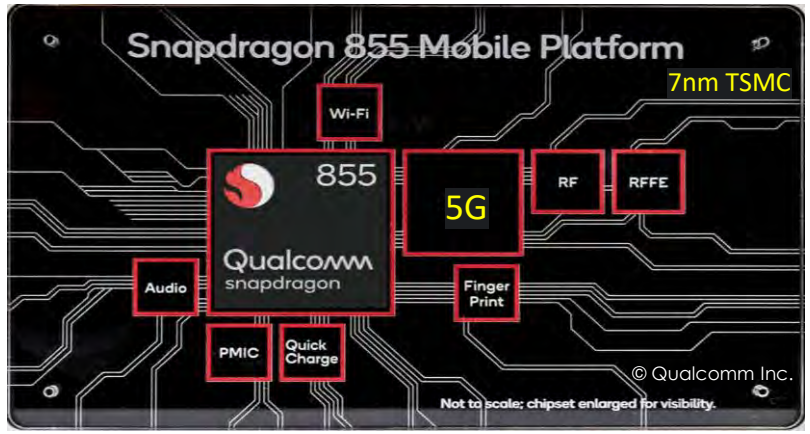
- NVIDIA Jetson AGX Xavier:
- High-performance system on a chip for autonomous machines
- Heterogeneous SoC contains:
  - Eight-core 64-bit ARMv8.2 CPU cluster (Carmel)
  - 1.4 CUDA TFLOPS (FP32) GPU with additional inference optimizations (Volta)
  - 11.4 DL TOPS (INT8) Deep learning accelerator (NVDLA)
  - 1.7 CV TOPS (INT8) 7-slot VLIW dual-processor Vision accelerator (PVA)
  - A set of multimedia accelerators (stereo, LDC, optical flow)
- Provides researchers access to advanced high-performance SOC environment







Qualcomm 855 SoC (SM8510P)



Kyro 485 (8-ARM Prime+BigLittle Cores)

Hexagon 690 (DSP + AI)

- Quad threaded Scalar Core
- DSP + 4 Hexagon Vector Xccelerators
- New Tensor Xccelerator for AI
- Apps: AI, Voice Assistance, AV codecs

Adreno 640

- Vulkan, OpenCL, OpenGL ES 3.1
- Apps: HDR10+, HEVC, Dolby, etc
- Enables 8k-360° VR video playback
- 20% faster compared to Adreno 630

Connectivity (5G)

- Snapdragon X24 LTE (855 built-in) modem LTE Category 20
- Snapdragon X50 5G (external) modem (for 5G devices)
- Qualcomm Wi-Fi 6-ready mobile platform: (802.11ax-ready, 802.11ac Wave 2, 802.11ay, 802.11ad)
- Qualcomm 60 GHz Wi-Fi mobile platform: (802.11ay, 802.11ad)
- Bluetooth Version: 5.0
- Bluetooth Speed: 2 Mbps
- High accuracy location with dual-frequency GNSS.

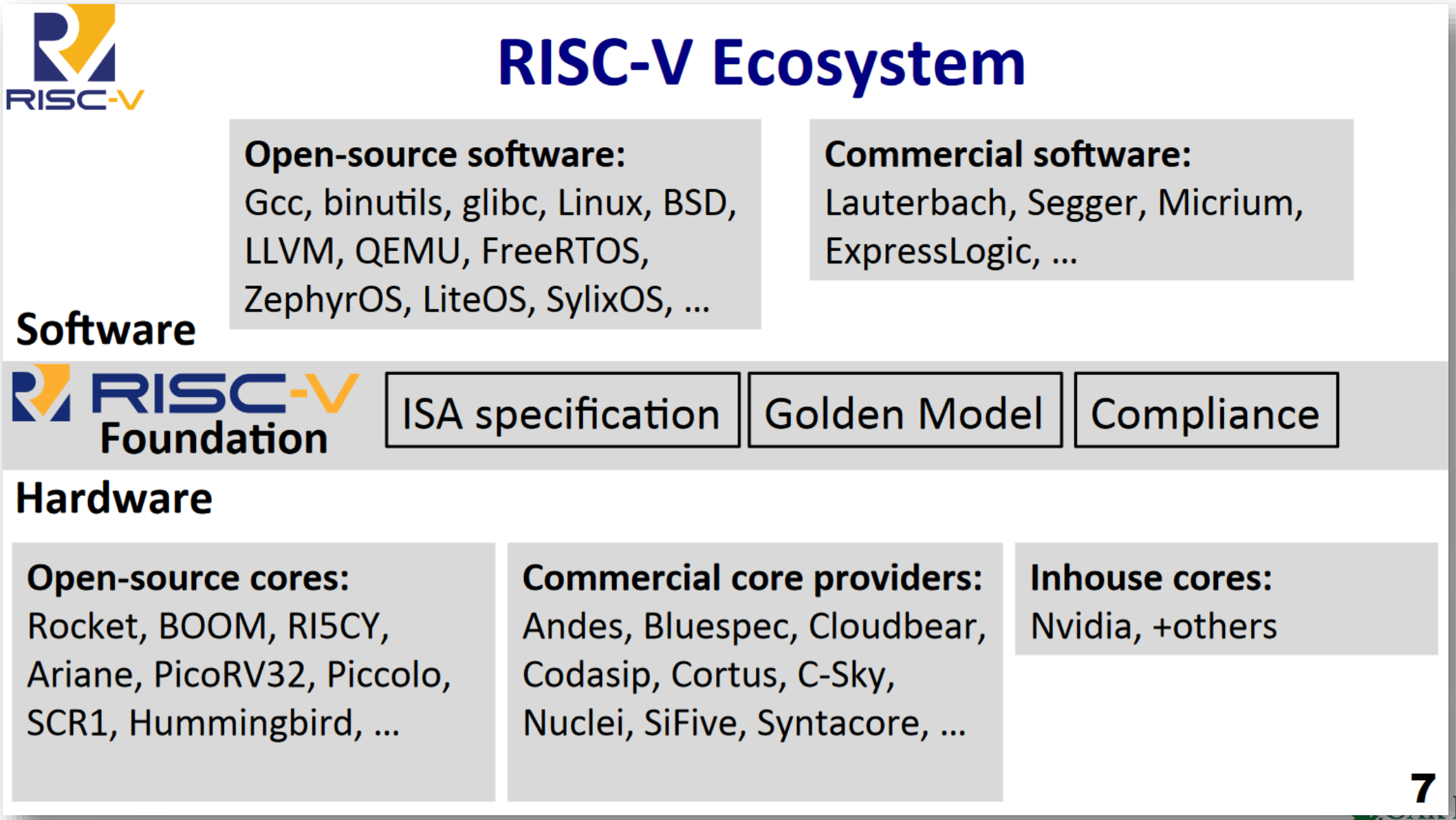
Spectra 360 ISP

- New dedicated Image Signal Processor (ISP)
- Dual 14-bit CV-ISPs; 48MP @ 30fps single camera
- Hardware CV for object detection, tracking, streo depth process
- 6DoF XR Body tracking, H265, 4K60 HDR video capture, etc.

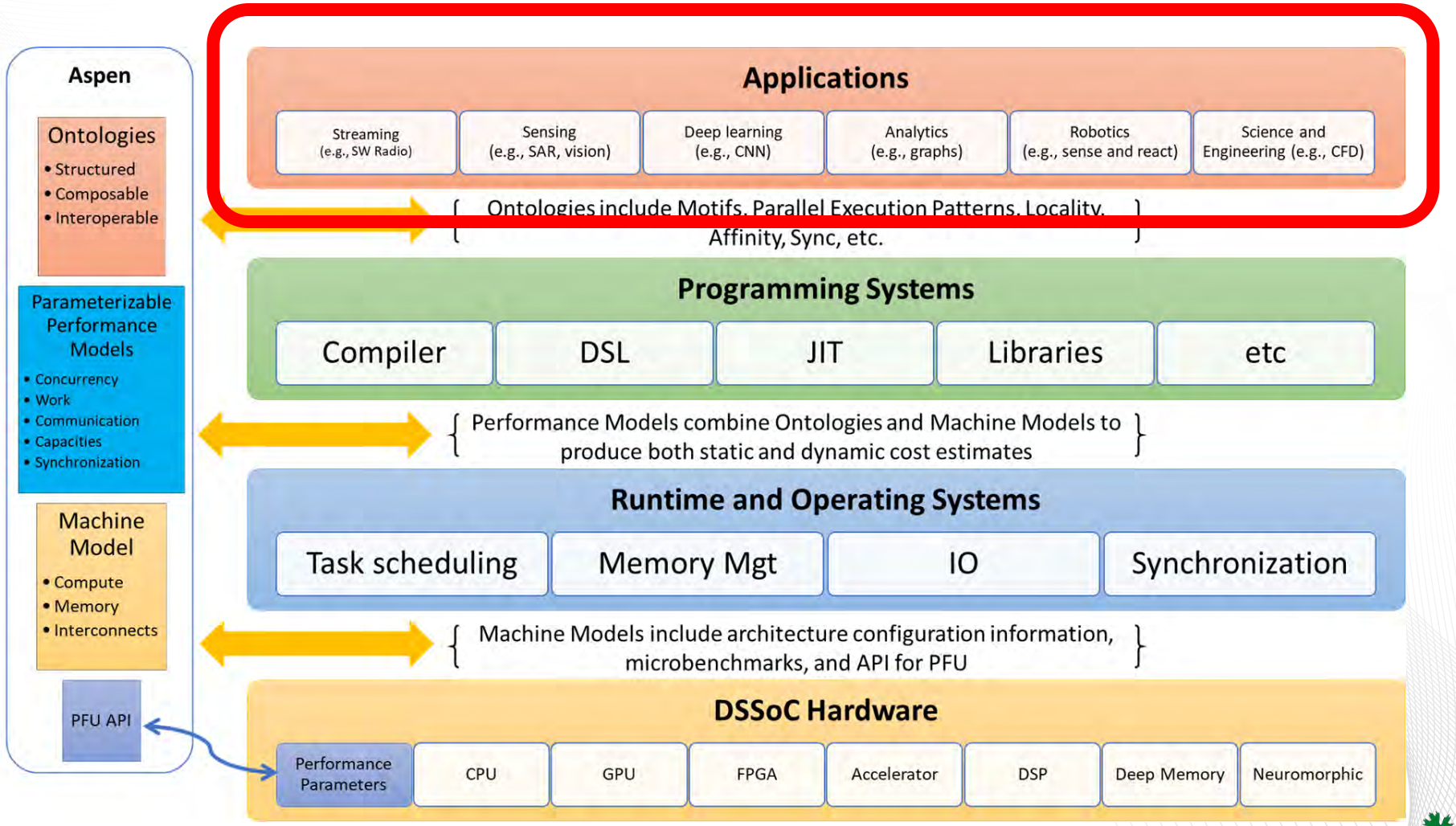
Qualcomm Development Board connected to (mcmurdo) HPZ820

- Connected Qualcomm board to HPZ820 through USB
- Development Environment: Android SDK/NDK
- Login to mcmurdo machine
  $ ssh –Y mcmurdo
- Setup Android platform tools and development environment
  $ source /home/nqx/setup_android.source
- Run Hello-world on ARM cores
  $ git clone https://code.ornl.gov/nqx/helloworld-android
  $ make compile push run
- Run OpenCL example on GPU
  $ git clone https://code.ornl.gov/nqx/opencl-img-processing
  - Run Sobel edge detection
    $ make compile push run fetch
- Login to Qualcomm development board shell
  $ adb shell
  $ cd /data/local/tmp

RISC-V


RISC-V Ecosystem
Open-source software:
Gcc, binutils, glibc, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS, …
Commercial software:
Lauterbach, Segger, Micrium, ExpressLogic, …
Software
RISC-V Foundation
ISA specification
Golden Model
Compliance
Hardware
Open-source cores:
Rocket, BOOM, RI5CY, Ariane, PicoRV32, Piccolo, SCR1, Hummingbird, …
Commercial core providers:
Andes, Bluespec, Cloudbear, Codasip, Cortus, C-Sky, Nuclei, SiFive, Syntacore, …
Inhouse cores:
Nvidia, +others
7

Applications



UNITED STATES FREQUENCY ALLOCATIONS

THE RADIO SPECTRUM

RADIO SERVICES COLOR LEGEND

- AERONAUTICAL MOBILE
- AERONAUTICAL MOBILE SATELLITE
- AERONAUTICAL RADIONAVIGATION
- AMATEUR
- AMATEUR SATELLITE
- BROADCASTING
- BROADCASTING SATELLITE
- EARTH EXPLORATION SATELLITE
- FIXED
- FIXED SATELLITE
- INTER-SATELLITE
- LAND MOBILE
- LAND MOBILE SATELLITE
- MARITIME MOBILE
- MARITIME MOBILE SATELLITE
- MARITIME RADIONAVIGATION
- METEOROLOGICAL
- METEOROLOGICAL SATELLITE
- MOBILE
- MOBILE SATELLITE
- RADIO ASTRONOMY
- RADIODETERMINATION SATELLITE
- RADIOLOCATION
- RADIOLOCATION SATELLITE
- RADIONAVIGATION
- RADIONAVIGATION SATELLITE
- SPACE OPERATION
- SPACE RESEARCH
- STANDARD FREQUENCY AND TIME SIGNAL
- STANDARD FREQUENCY AND TIME SIGNAL SATELLITE

ACTIVITY CODE

- FEDERAL EXCLUSIVE
- FEDERAL/NON-FEDERAL SHARED
- NON-FEDERAL EXCLUSIVE

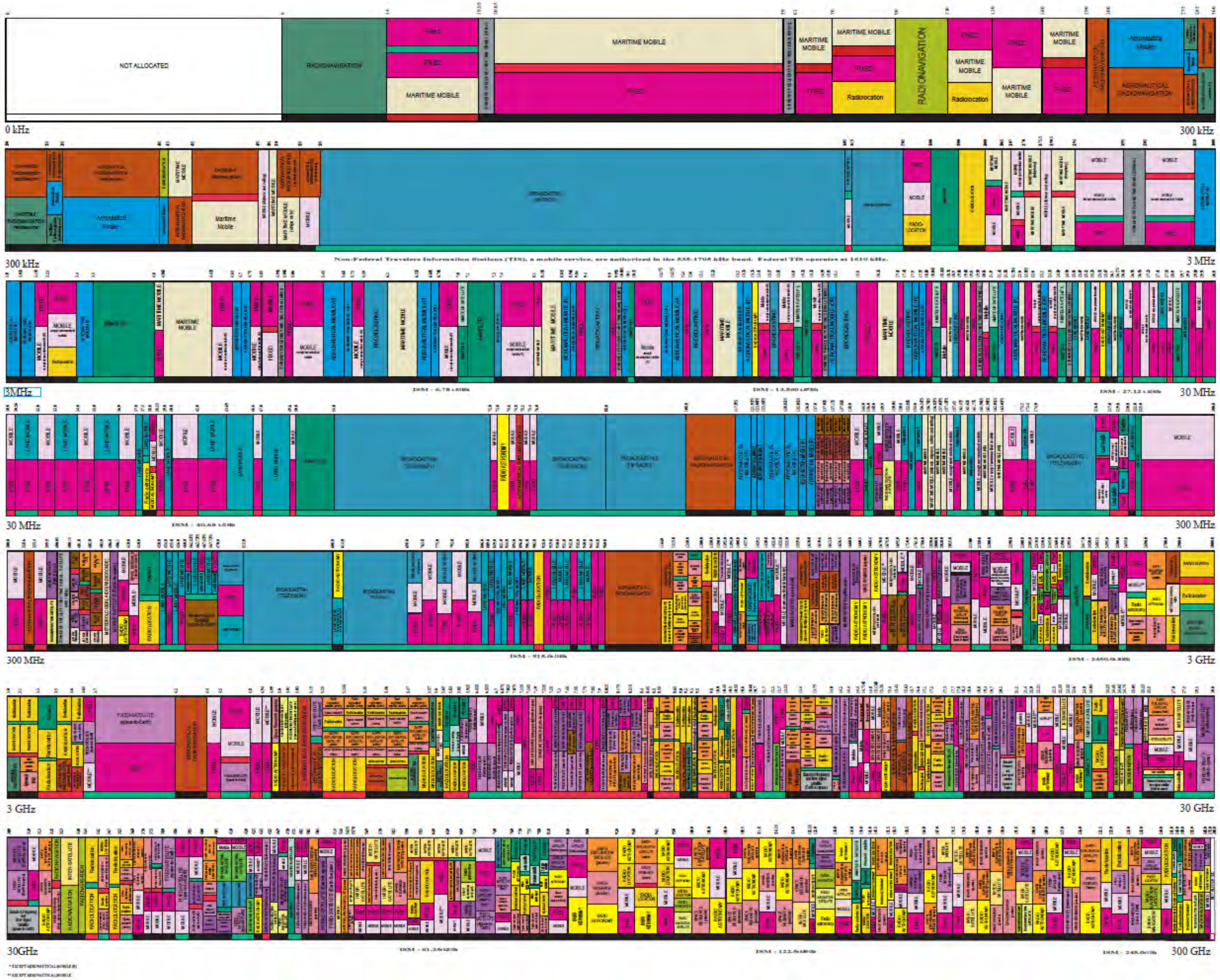
ALLOCATION USAGE DESIGNATION

SERVICE	EXAMPLE	DESCRIPTION
Primary	FIXED	Capital Letters
Secondary	Mobile	1st Capital with lower case letters

This chart is a graphic single-point-in-time portrayal of the Table of Frequency Allocations used by the FCC and NTIA. As such, it may not completely reflect all aspects, i.e. footnotes and recent changes made to the Table of Frequency Allocations. Therefore, for complete information, users should consult the Table to determine the current status of U.S. allocations.

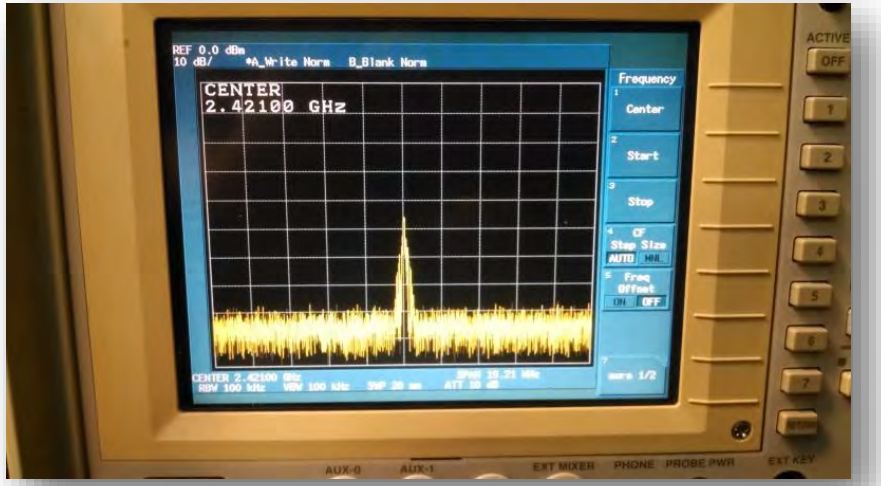
U.S. DEPARTMENT OF COMMERCE
National Telecommunications and Information Administration
Office of Spectrum Management

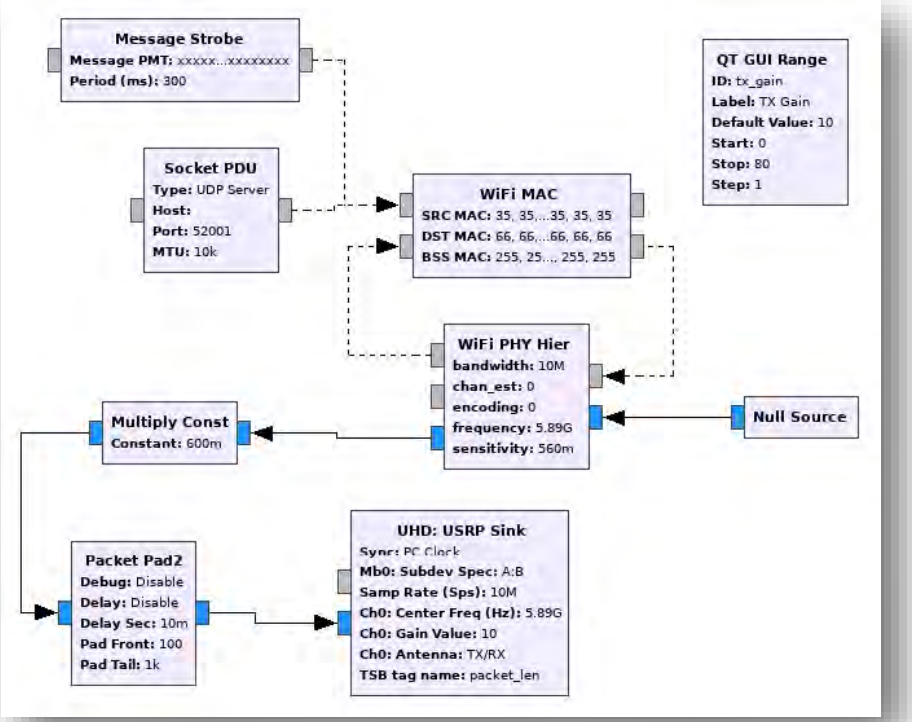
JANUARY 2016



Software Defined Radio

- A radio communication system where components that have been traditionally implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a computer, phone, or embedded system (Wikipedia).
- Gnu Radio – open software for SDR
- Composable modules and workflows for signal processing



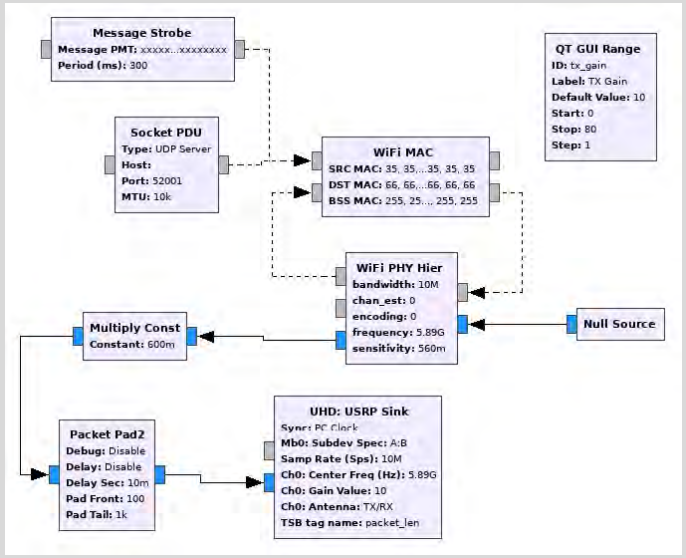


End-to-End System: Gnu Radio for Wifi on two NVIDIA Xavier SoCs

Xavier SoC

Video/Image Files

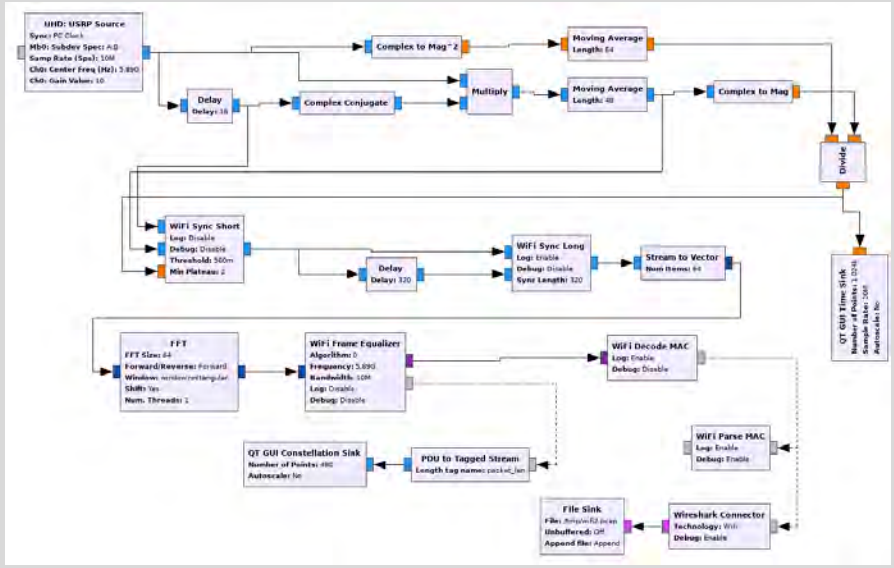
UDP



GNR IEEE-802.11 Transmit (TX)

Antenna

Xavier SoC



IEEE-802.11 Receive (RX)

UDP

OpenCV

OpenCL

- Signal processing: An open-source implementation of IEEE-802.11 WIFI a/b/g with GNR OOT modules.
- Input / Output file support via Socket PDU (UDP server) blocks
- Image/Video transcoding with OpenCL/OpenCV

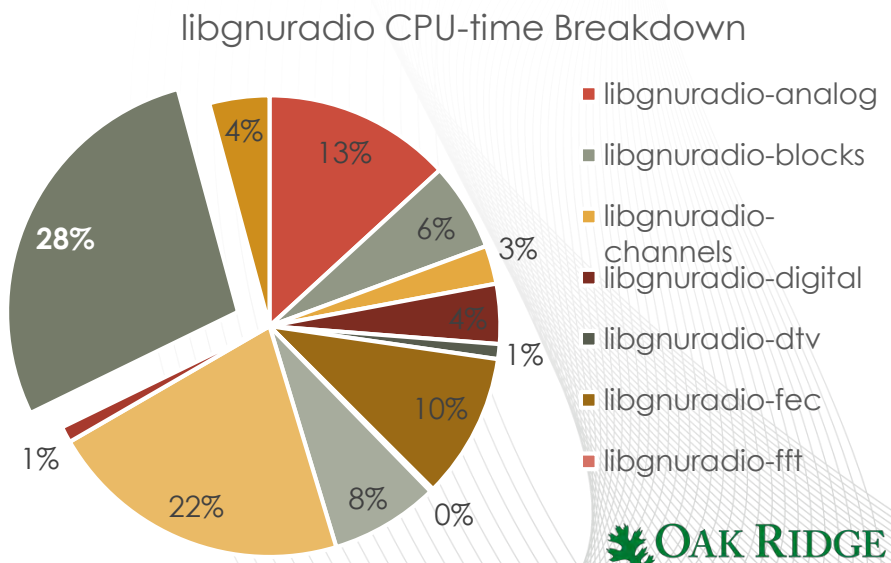
RF + SoC Hardware Testbed

- GNU Radio ➔ 2x Ettus B210: [RF-A: Loopback cable, RF-B: Antennas, Wifi Frequency: 5.89 GHz]

Applications Summary

- **Preliminary SDR Application Profiling:**
  - **Created fully automated GRC profiling toolkit**
  - Ran each of the 89 flowgraph for 30 seconds
  - Profiled with performance counters
  - Major overheads:
    - Python glue code (libpython), O/S threading & profiling (kernel.kallsyms, libpthread), libc, ld, Qt
  - Runtime overhead:
    - Will require significant consideration when run on SoC
    - Cannot be executed in parallel
    - Hardware assisted scheduling is essential

Library	Percentage
[kernel.kallsyms]	27.8547
libpython	18.6281
libgnuradio	**11.7548**
libc	7.7503
ld	3.8839
libvolk	**3.7963**
libperl	3.7837
[unknown]	3.6465
libQt5	2.9866
libpthread	2.1449



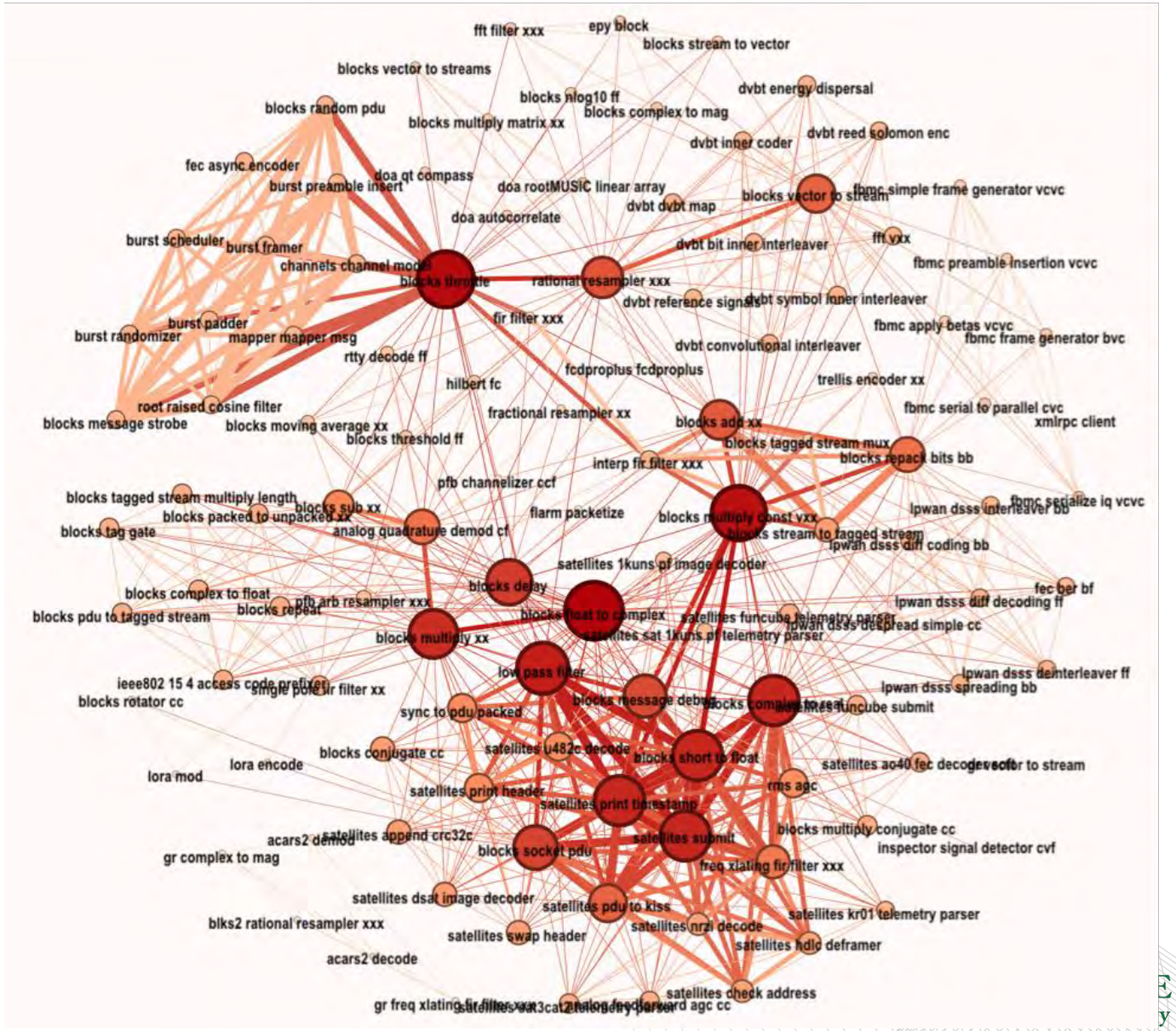
OAK RIDGE National Laboratory

GRC statistics: Block Proximity Analysis

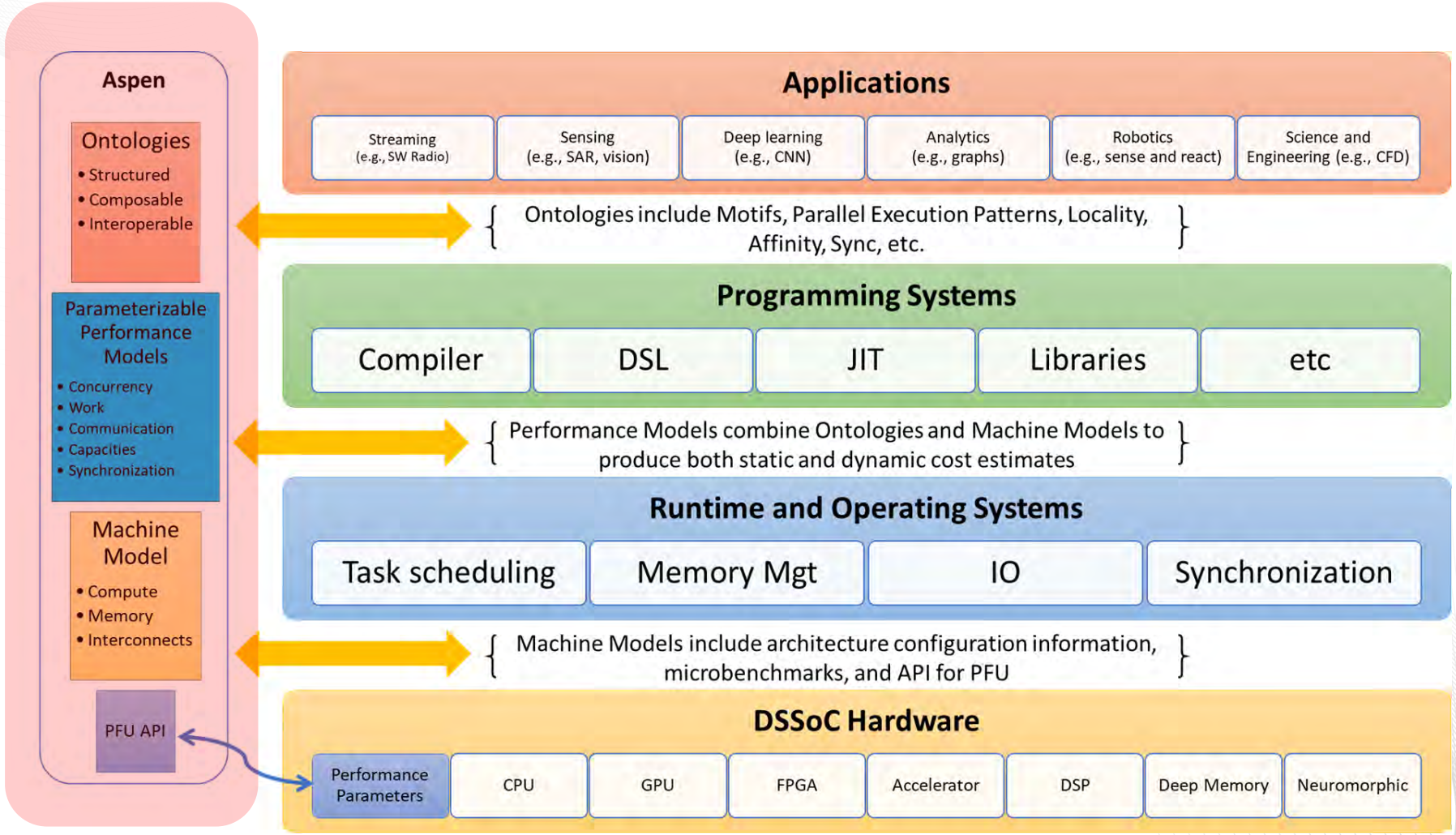
Block proximity analysis

- Creates a graph:
  - Nodes: Unique block types
  - Edges: Blocks used in the same GRC file.
  - Every co-occurrence increases edge weight by 1.
- This example was run
  - With `--mode proximityGraph`
  - On randomly selected sub-set of GRC files

borip-USRP-UHD.grc	live_signal_detection.grc
cdma_tx_hier1.grc	psk_burst_ldpc_tx.grc
cdma_tx_hier.grc	psk_burst_tx.grc
dsat.grc	rfnoc_digital_gain_network_host.grc
dsss_sim_perfekt_sync_fg_without_fec.grc	rtty_decode.grc
dvbt_tx_demo_8k_QPSK_rate78.grc	run_RootMUSIC_lin_array_simulation.grc
fbmc_frame_generator_perf_test.grc	sat_1kuns_pf.grc
flarm_2chan.grc	sat_3cat_2.grc
frontend_lilacsat1_rx_fcdpp.grc	snapshot-approach.grc
fsk_tx.grc	symbol_differential_filter_phases.grc
ieee802_15_4_OQPSK_PHY.grc	symbol_sampling.grc
jy1sat.grc	tx_usrp.grc
kr01.grc	usrp-input.grc



Integrating Modeling Across the Stack with Aspen



Aspen: Abstract Scalable Performance Engineering Notation

Model Creation

- Static analysis via compiler, tools
- Empirical, Historical
- Manual (for future applications)

Representation in Aspen

- **Modular**
- **Sharable**
- **Composable**
- **Reflects prog structure**

Model Uses

- Interactive tools for graphs, queries
- Design space exploration
- Workload Generation
- Feedback to Runtime Systems

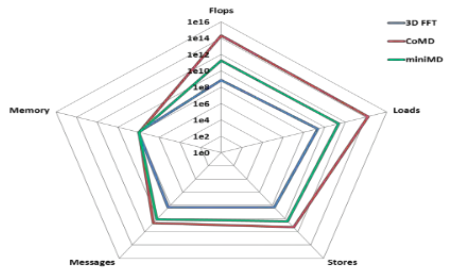
Source code

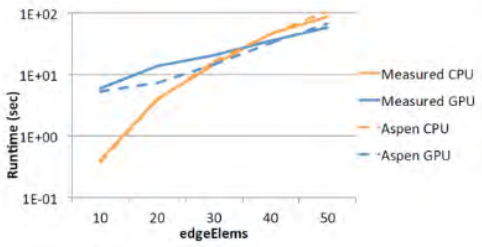
```
static inline
void CalcMonotonicQGradientsForElems(Index_t p_nodelist[T_NUMELEM8],
        Real_t p_x[T_NUMNODE], Real_t p_y[T_NUMNODE], Real_t p_z[T_NUMNODE],
        Real_t p_xd[T_NUMNODE], Real_t p_yd[T_NUMNODE],Real_t p_zd[T_NUMNODE],
        Real_t p_volo[T_NUMELEM], Real_t p_vnew[T_NUMELEM],
        Real_t p_delx_zeta[T_NUMELEM], Real_t p_delv_zeta[T_NUMELEM],
        Real_t p_delx_xi[T_NUMELEM], Real_t p_delv_xi[T_NUMELEM],
        Real_t p_delx_eta[T_NUMELEM], Real_t p_delv_eta[T_NUMELEM])
{
   Index_t i;
   Index_t numElem = m_numElem;
#pragma acc parallel loop independent present(p_vnew, p_nodelist, p_x, p_y, p_z, p_xd, \
p_yd, p_zd, p_volo, p_delx_xi, p_delx_eta, p_delx_zeta, p_delv_xi, p_delv_eta,\
p_delv_zeta)
   for (i = 0 ; i < numElem ; ++i ) {
      const Real_t ptiny = 1.e-36 ;
      Real_t ax,ay,az ;
      Real_t dxv,dyv,dzv ;

      const Index_t *elemToNode = &p_nodelist[8*i];
      Index_t n0 = elemToNode[0] ;
      Index_t n1 = elemToNode[1] ;
      Index_t n2 = elemToNode[2] ;
      Index_t n3 = elemToNode[3] ;
      Index_t n4 = elemToNode[4] ;
      Index_t n5 = elemToNode[5] ;
      Index_t n6 = elemToNode[6] ;
      Index_t n7 = elemToNode[7] ;

      Real_t x0 = p_x[n0] ;
```

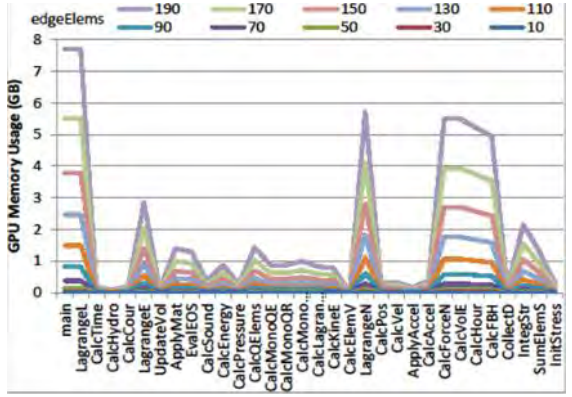
E.g., MD, UHPC CP 1, Lulesh, 3D FFT, CoMD, VPFFT, …

Aspen code

```
kernel CalcMonotonicQGradients {
  execute [numElems]
  {
    loads [8 * indexWordSize] from nodelist
    // Load and cache position and velocity.
    loads/caching [8 * wordSize] from x
    loads/caching [8 * wordSize] from y
    loads/caching [8 * wordSize] from z

    loads/caching [8 * wordSize] from xvel
    loads/caching [8 * wordSize] from yvel
    loads/caching [8 * wordSize] from zvel

    loads [wordSize] from volo
    loads [wordSize] from vnew
    // dx, dy, etc.
    flops [90] as dp, simd
    // delvk delxk
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_xeta
    // delxi delvi
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delx_xi
    // delxj and delvj
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_eta
  }
}
```

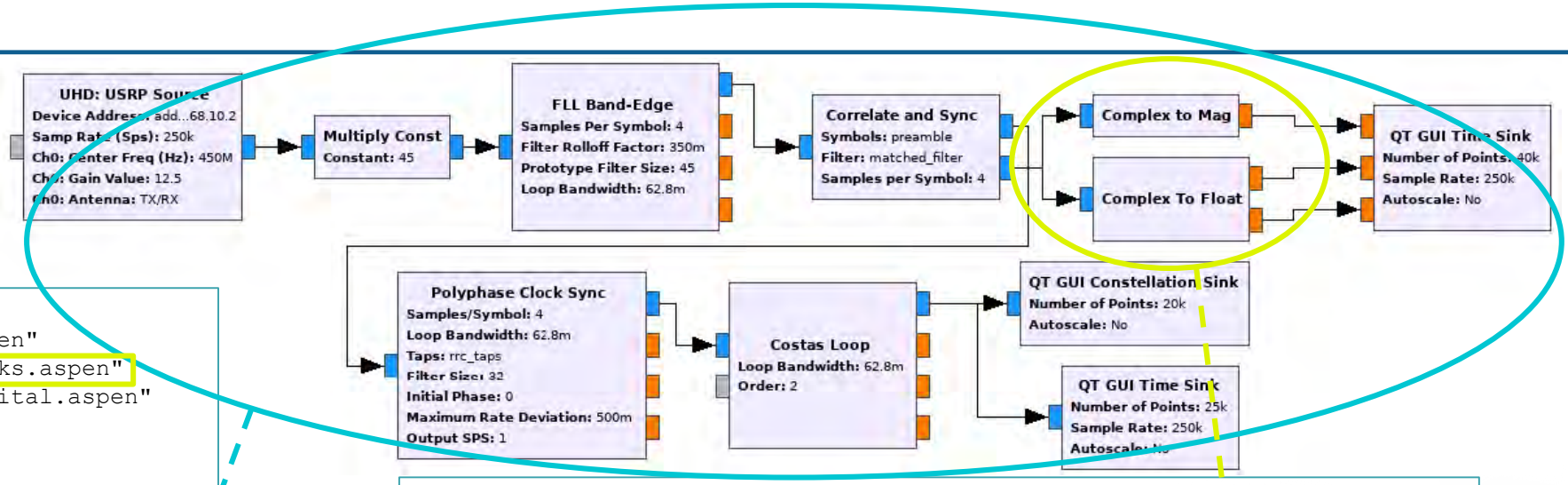






K. Spafford, et al, "Aspen: A Domain Specific Language for Performance Modeling," in Proc. *SC12.*

GNURadio Flowgraph to Aspen Application Model Conversion




```
model demod-uhd-sync{
  import uhd from "gr-uhd.aspen"
  import blocks from "gr-blocks.aspen"
  import digital from "gr-digital.aspen"

  kernel main {
    call uhd.usrp_source()
    call blocks.multiply_const_vxx()
    call digital.fll_band_edge_cc()
    call digital.correlate_and_sync_cc()

    parallel {
      sequence{
        parallel {
          call blocks.complex_to_mag()
          call blocks.complex_to_float()
        }
        call blocks_file_sink()
      }
      sequence{
        call digital.pfb_clock_sync_xxx()
        call digital.costas_loop_cc()
        parallel {
          call blocks.file_sink()
          call blocks.file_sink()
        }
      }
    }
  }
}
```

demod-uhd-sync.aspen

```
model blocks {
  kernel complex_to_mag {
    param aspen_param_default = 1
    param aspen_param_sizeof_float = 4
    param noutput_items = 8192
    param aspen_param_sizeof_FComplex = 8
    execute [noutput_items] "block_clComplexToMag_kernel4"
      flops [1] as integer
    execute  "block_clComplexToMag_kernel4__intracommIN"
      intracomm [(aspen_param_sizeof_FComplex*noutput_items)] as copyin
    map [noutput_items] "mapblock_clComplexToMag_kernel4"
      execute  "block_clComplexToMag_kernel7" {
        loads [(1*aspen_param_sizeof_FComplex)] as stride(1)
        loads [(1*aspen_param_sizeof_FComplex)] as stride(1)
        stores [(1*aspen_param_sizeof_float)] as stride(1)
        flops [4] as dp, simd
      }
    execute  "block_clComplexToMag_kernel4__intracommOUT"
      intracomm [(aspen_param_sizeof_float*noutput_items)] as copyout
  }
  kernel complex_to_float {...}
  ...
}
```

gr-blocks.aspen

Graph-Based Abstract Machine Model

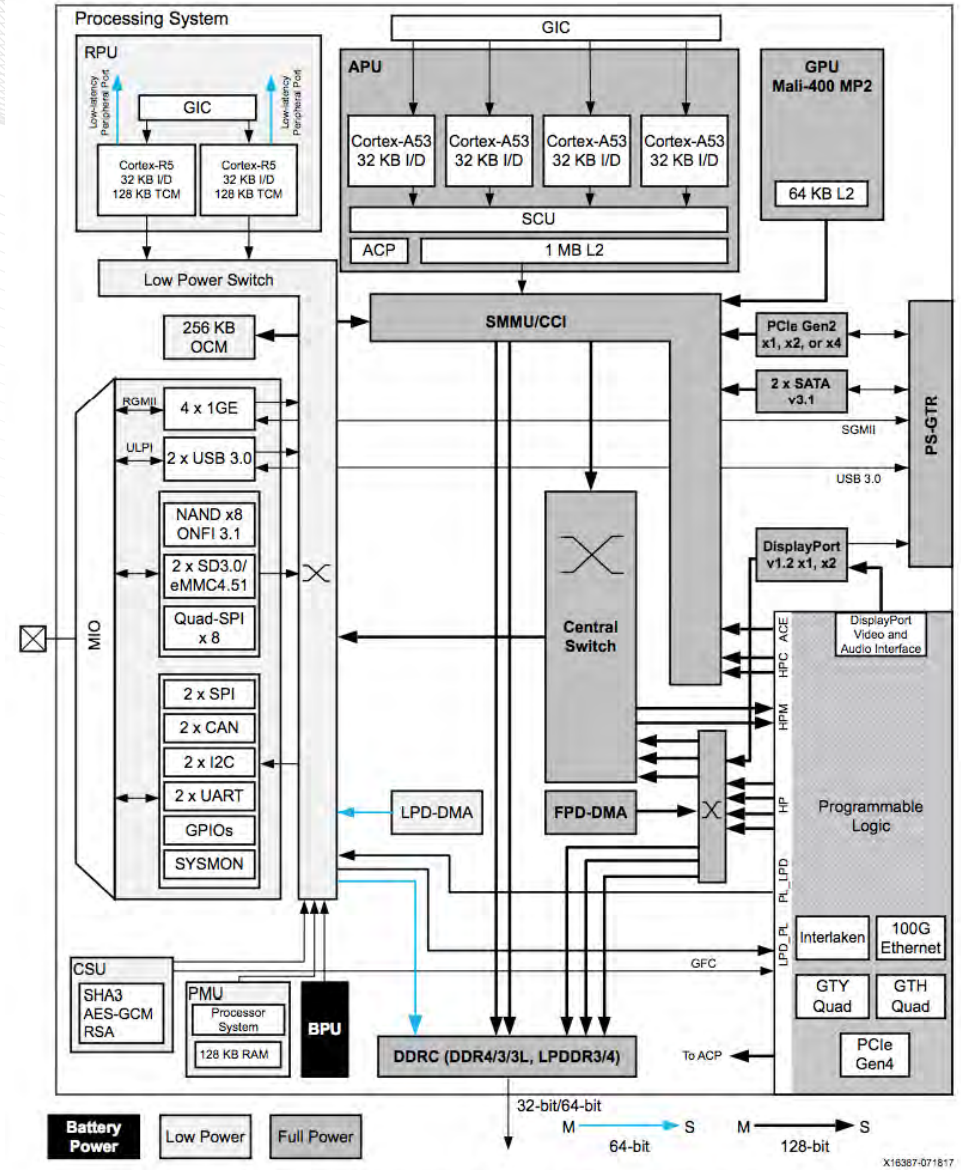


Figure 3-1: **Zynq UltraScale+ MPSoC Top-Level Block Diagram**

```
class Zynq::Board-ZCU102 : Aspen::CompoundNode {
// Processing units
  Zynq::APU cpu;
  ARM::Mali400MP2 gpu;
  ARM::CortexR5 rpu;
  Xilinx::UltraScale+<nFPUs=400M> fpga;

  // Memory
  Aspen::DDR3<freq=2000MHZ, CL=16> systemMemory;

  // Memory controllers, switchs, mmus
  Zynq::SMMU smmu;
  Aspen::Switch<bw=100GBs, latency= 25ns> lpSwitch;
  Aspen::Switch<bw=1TBs, latency= 35ns> centralSwitch;
  Aspen::PCIController<ver=3, totalLanes=24> pciController;

  // Define interconnects (edges)
  Aspen::Bus<bw=400GBps> cci_fp;
  Aspen::Bus<bw=100GBps> cci_lp;
  Aspen::PCIe<version=3,lane=16> pcieBus;

  @add
    cpu --cci_fp-- smmu;
    gpu --cci_fp-- smmu;
    fpga --cci_fp-- smmul
    systemMemory --cci_fp[2]-- smmu; // Multiple links

    smmu --cci_fp-- centralSwitch;
    smmu --cci_fp-- pciController;
    fpga --cci_fp[2]-- centralSwitch

    lpSwitch --cci_fp-->> smmu; // Unidirectional link
    lpSwitch <<--cci_fp-- centralSwitch
    rpu --cci_lp[2]-- lpSwitch;

    pciController --pcieBus -->> Aspen::OUTPUT
    pciController <<--pcieBus -->> Aspen::INPUT
```

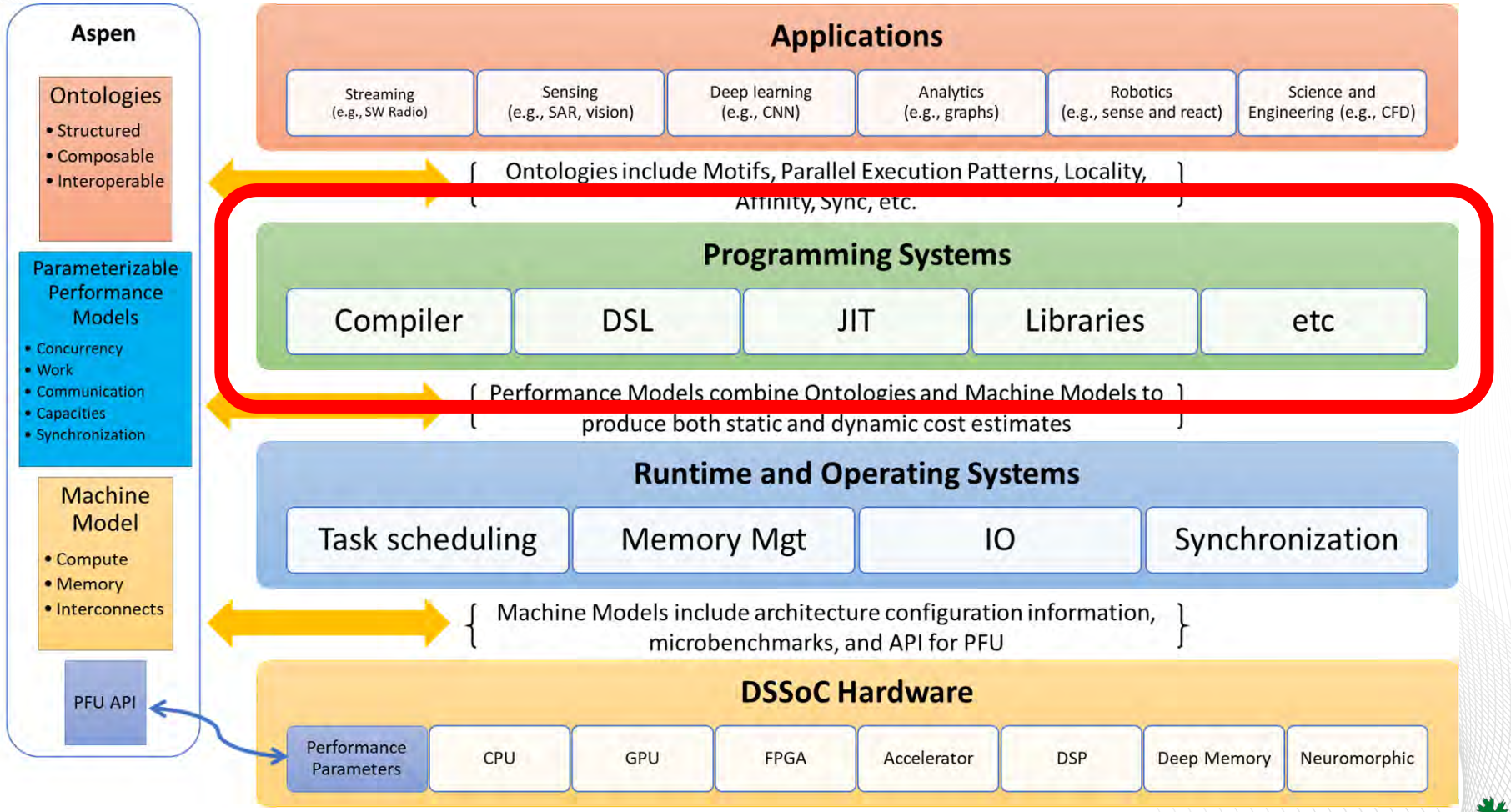
Nodes

Edges

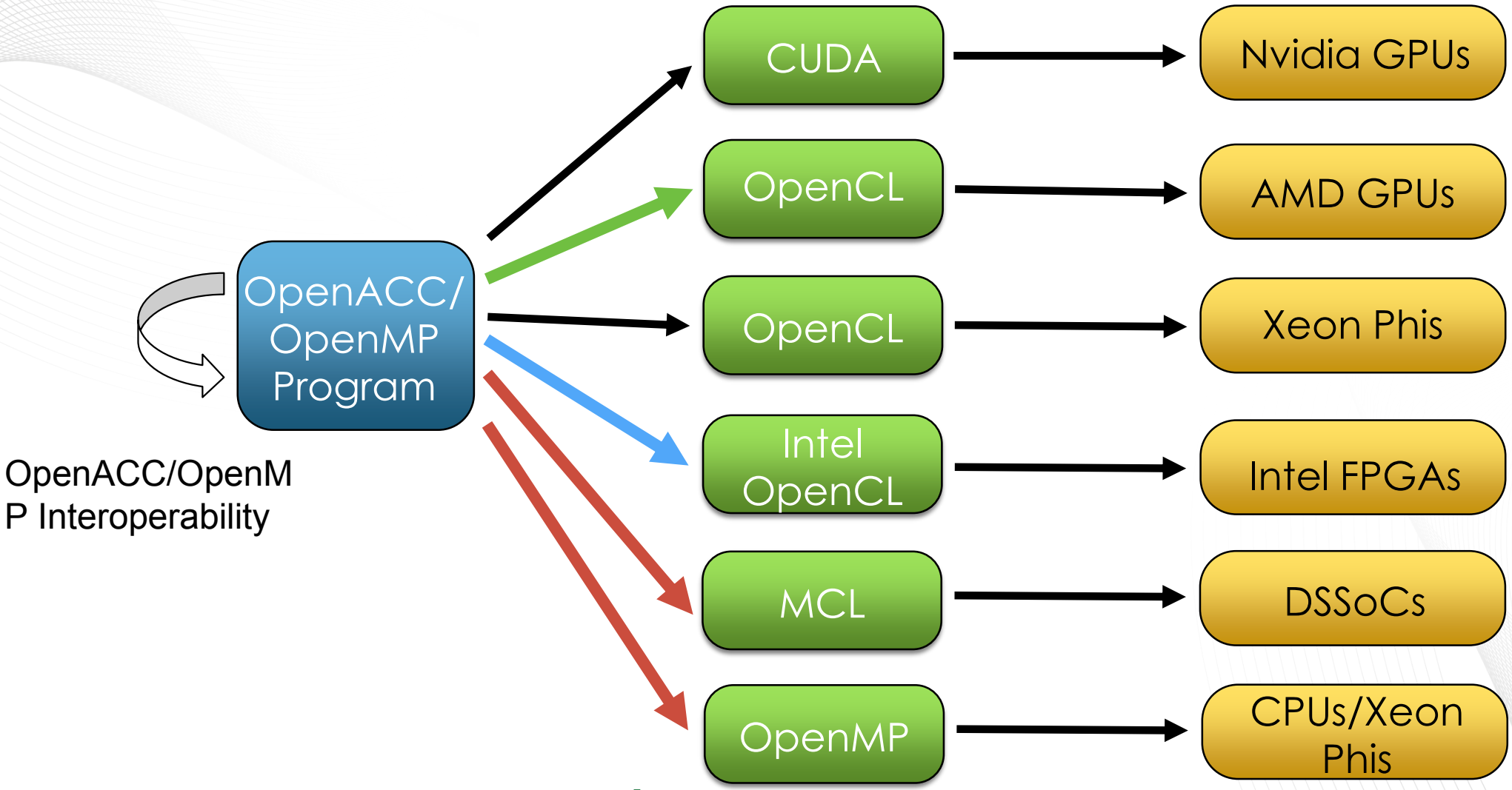
Graph

Belviranli, M E, et al, "FLAME: Graph-based Hardware Representations for Rapid and Precise Performance Modeling", DATE'19

Programming Systems



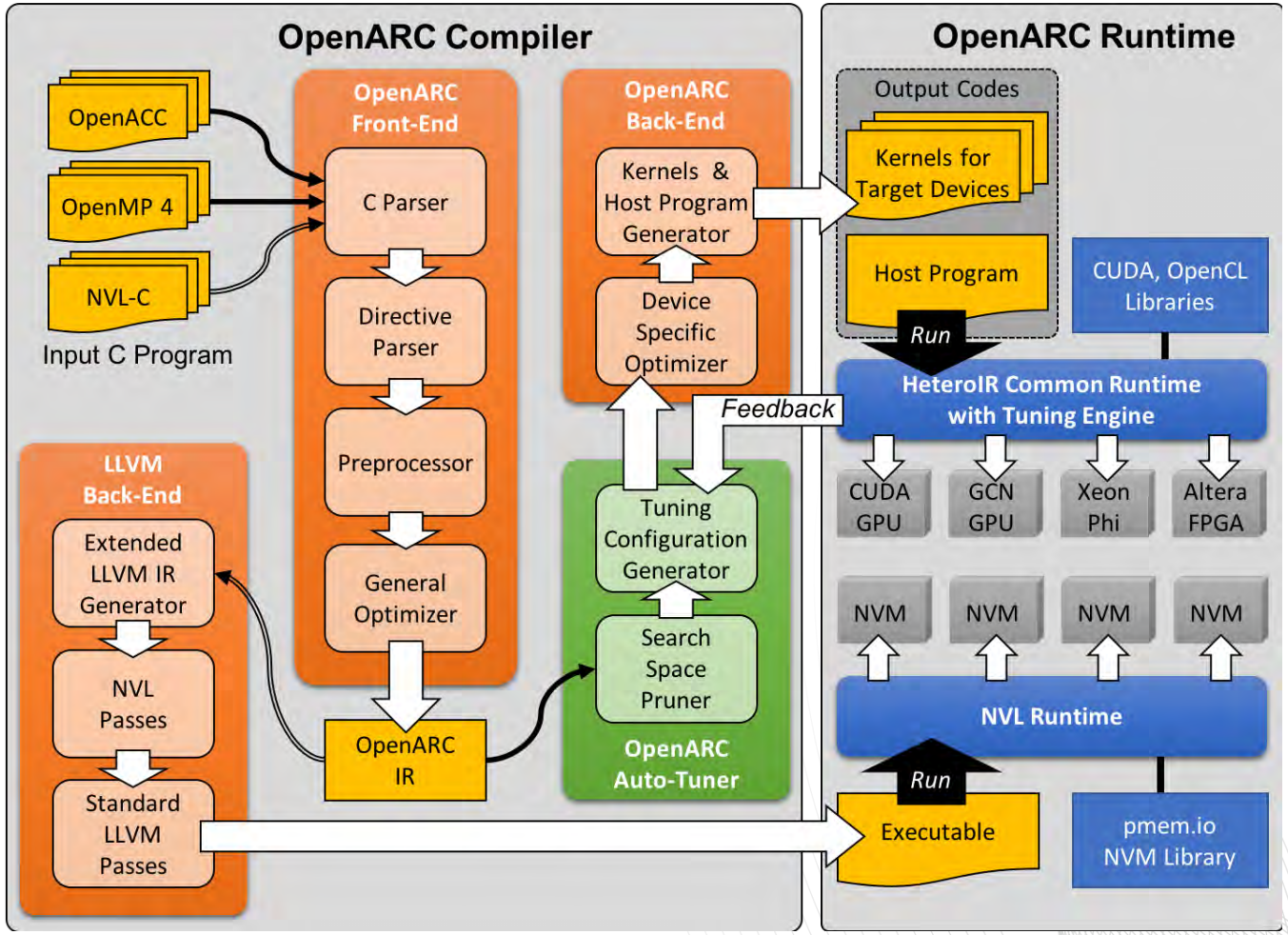
Heterogeneous Architecture Support by OpenARC



OpenACC/OpenM
P Interoperability

OpenARC: Open Accelerator Research Compiler*

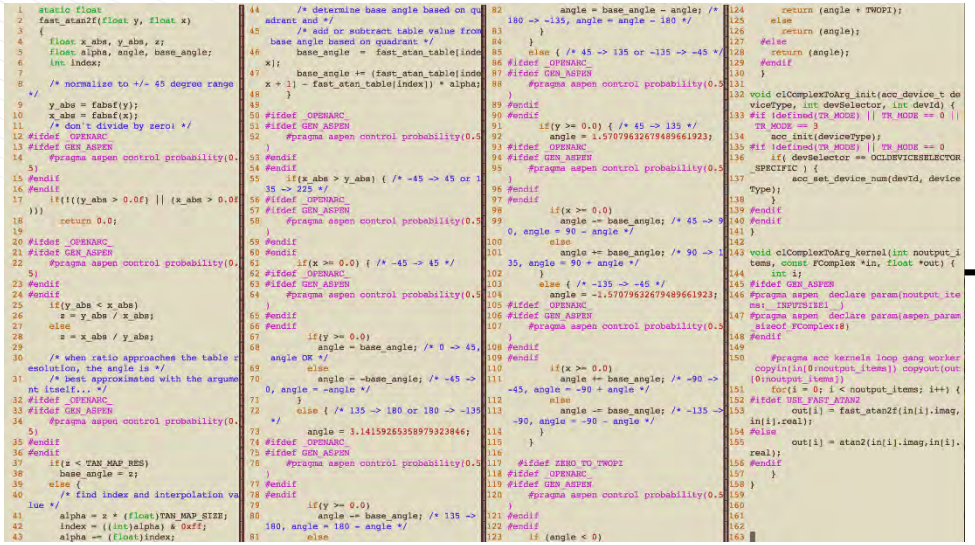
- OpenARC is the first open-sourced, OpenACC/OpenMP compiler supporting Altera FPGAs, in addition to NVIDIA/AMD GPUs and Intel Xeon Phis.
- OpenARC is a high-level intermediate representation based, extensible compiler framework, where various performance optimizations, traceability mechanisms, fault tolerance techniques, etc., can be built for the complex heterogeneous computing.



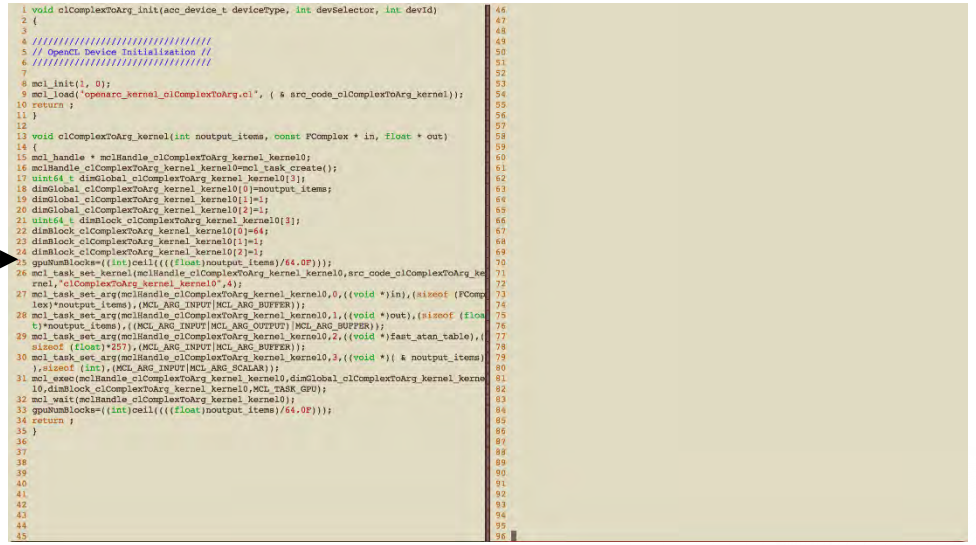
* OpenARC, Lee, HPDC '14.

Example Translation of Gnu Radio Module: OpenACC to MCL (Targeting General Heterogeneous Devices)

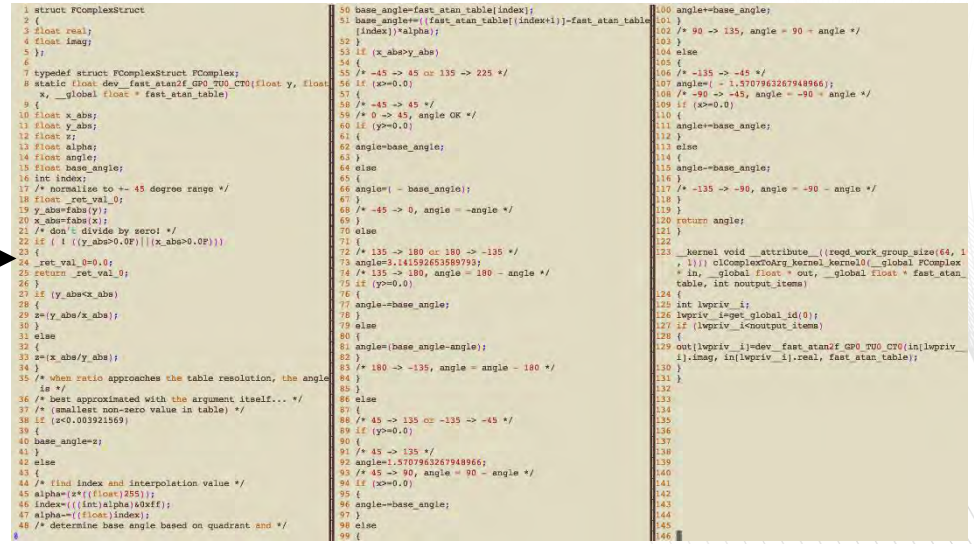
Input OpenACC code



Output MCL host code

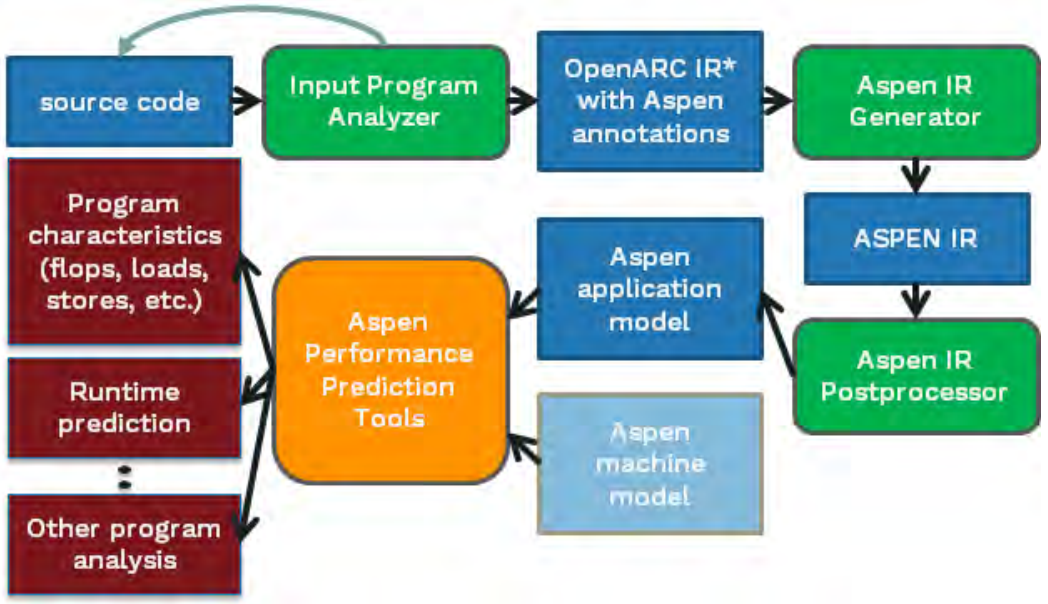


Output MCL kernel code

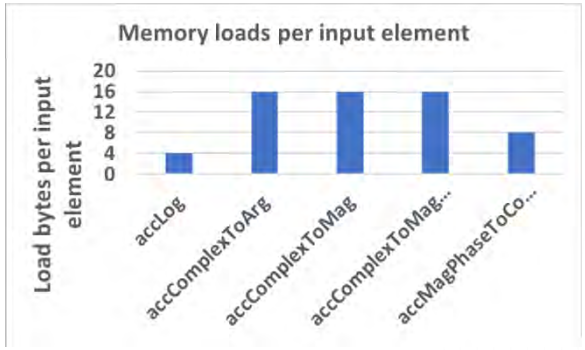


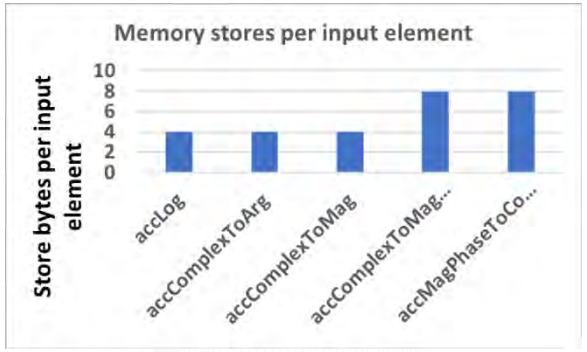
Program Characteristics Prediction of GNU Radio Blocks using Aspen

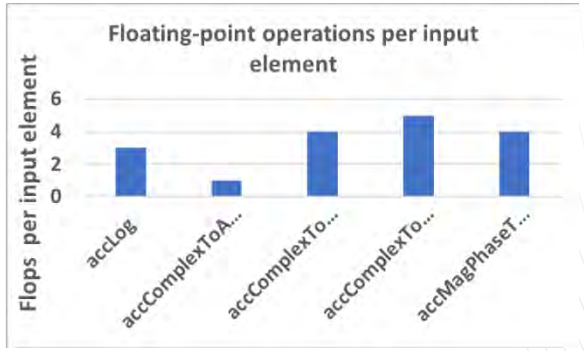
- OpenARC automatically generates a structured Aspen performance model from the ported OpenACC code of the GNU Radio blocks.
- Aspen performance prediction tools digest the generated Aspen models and derive performance predictions for the target application.

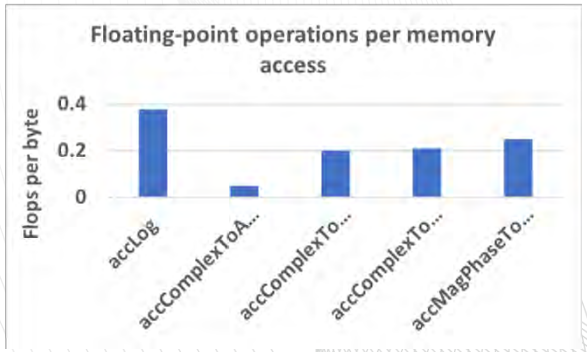


COMPASS: A Framework for Automated Performance Modeling and Prediction









Example Translation of Gnu Radio Module: OpenACC to Aspen

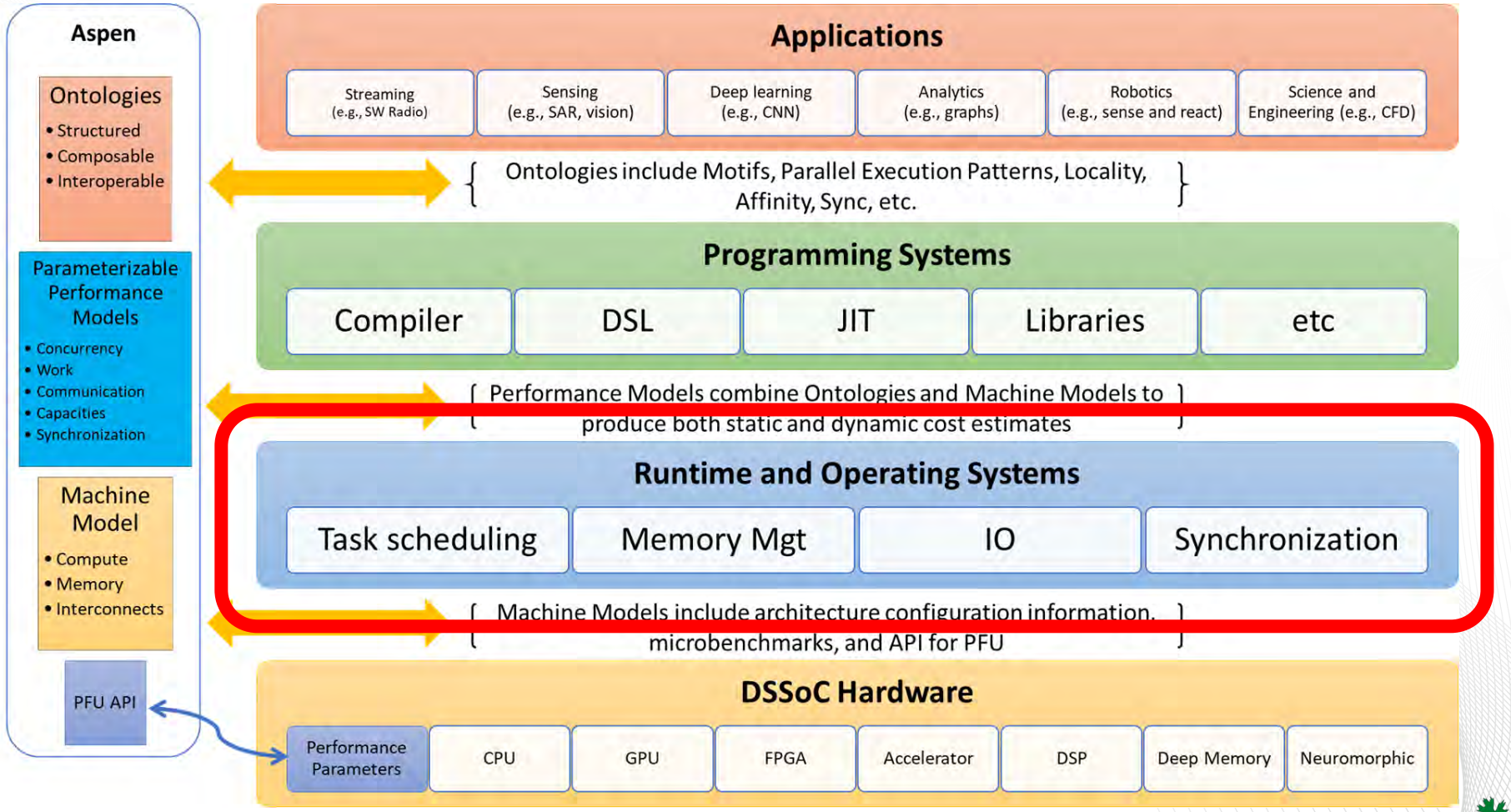
Input OpenACC code

```
static float
fast_atan2f(float y, float x)
{
  float x_abs, y_abs, z;
  float alpha, angle, base_angle;
  int index;

  /* normalize to +/- 45 degree range */
  y_abs = fabsf(y);
  x_abs = fabsf(x);
  /* don't divide by zero! */
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
  if(!((y_abs > 0.0f) || (x_abs > 0.0f)))
    return 0.0;

#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
  if(y_abs < x_abs)
    z = y_abs / x_abs;
  else
    z = x_abs / y_abs;

  /* when ratio approaches the table resolution, the angle is */
  /* best approximated with the argument itself... */
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
  if(z < TAN_MAP_RES)
    base_angle = z;
  else {
    /* find index and interpolation value */
    alpha = z * (float)TAN_MAP_SIZE;
    index = ((int)alpha) & 0xff;
    alpha -= (float)index;
    /* determine base angle based on quadrant and */
    /* add or subtract table value from base angle based on quadrant */
    base_angle = fast_atan_table[index];
    base_angle += (fast_atan_table[index + 1] - fast_atan_table[index]) * alpha;
  }

#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
  if(x_abs > y_abs) { /* -45 -> 45 or 135 -> 225 */
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
    if(x >= 0.0) { /* -45 -> 45 */
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
      if(y >= 0.0)
        angle = base_angle; /* 0 -> 45, angle OK */
      else
        angle = -base_angle; /* -45 -> 0, angle = -angle */
    }
    else { /* 135 -> 180 or 180 -> -135 */
      angle = 3.14159265358979323846;
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
      if(y >= 0.0)
        angle -= base_angle; /* 135 -> 180, angle = 180 - angle */
      else
        angle = base_angle - angle; /* 180 -> -135, angle = angle - 180 */
    }
  }
  else { /* 45 -> 135 or -135 -> -45 */
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
    if(y >= 0.0) { /* 45 -> 135 */
      angle = 1.57079632679489661923;
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
      if(x >= 0.0)
        angle -= base_angle; /* 45 -> 90, angle = 90 - angle */
      else
        angle += base_angle; /* 90 -> 135, angle = 90 + angle */
    }
    else { /* -135 -> -45 */
      angle = -1.57079632679489661923;
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
      if(x >= 0.0)
        angle += base_angle; /* -90 -> -45, angle = -90 + angle */
      else
        angle -= base_angle; /* -135 -> -90, angle = -90 - angle */
    }
  }
#ifdef ZERO_TO_TWOPI
#ifdef _OPENARC_
#ifdef GEN_ASPEN
    #pragma aspen control probability(0.5)
#endif
#endif
  if (angle < 0)
    return (angle + TWOPI);
  else
    return (angle);
#else
  return (angle);
#endif
}
```

Output Aspen Application Model

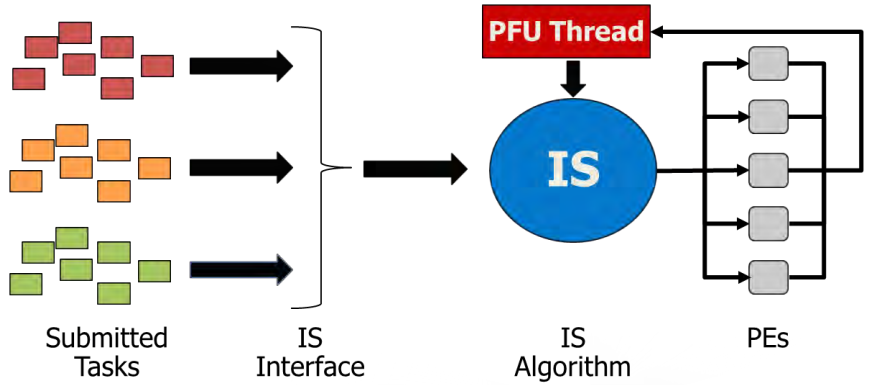
```
model clComplexToArg {
param aspen_param_default = 1
data fast_atan_table as Array(257, aspen_param_sizeof_float)
param aspen_param_sizeof_float = 4
param noutput_items = 8192
param aspen_param_sizeof_FComplex = 8
kernel fast_atan2f {
execute  "block_fast_atan2f19" {
flops [1] as sp
flops [1] as sp
flops [2] as sp
flops [1] as sp
}
probability {
[0.5] {
execute  "block_fast_atan2f26" {
flops [1] as sp
}
} else {
execute  "block_fast_atan2f27" {
flops [1] as sp
}
}
}
execute  "block_fast_atan2f29" {
flops [1] as dp
}
probability {
[(1-0.5)] {
execute  "block_fast_atan2f34" {
flops [1] as sp
flops [1] as sp
loads [(1*aspen_param_sizeof_float)] from fast_atan_table as stride(0)
loads [(2*aspen_param_sizeof_float)] from fast_atan_table as stride(0)
flops [3] as sp, simd
flops [1] as integer
}
}
execute  "block_fast_atan2f39" {
flops [1] as sp
}
probability {
[0.5] {
execute  "block_fast_atan2f41" {
flops [1] as dp
}
probability {
[0.5] {
execute  "block_fast_atan2f44" {
flops [1] as dp
}
} else {
execute  "block_fast_atan2f51" {
flops [1] as dp
}
probability {
[0.5] {
execute  "block_fast_atan2f54" {
flops [1] as sp
}
} else {
execute  "block_fast_atan2f55" {
flops [1] as sp
}
}
} else {
execute  "block_fast_atan2f56" {
flops [1] as dp
}
probability {
[0.5] {
execute  "block_fast_atan2f60" {
flops [1] as dp
}
probability {
[0.5] {
execute  "block_fast_atan2f63" {
flops [1] as sp
}
} else {
execute  "block_fast_atan2f64" {
flops [1] as sp
}
}
} else {
execute  "block_fast_atan2f67" {
flops [1] as sp
}
probability {
[0.5] {
execute  "block_fast_atan2f70" {
flops [1] as sp
}
} else {
execute  "block_fast_atan2f71" {
flops [1] as sp
}
}
}
kernel main {
execute [noutput_items] "block_clComplexToArg_kernel4" {
flops [1] as integer
}
execute  "block_clComplexToArg_kernel4__intracommIN" {
intracomm [(aspen_param_sizeof_FComplex*noutput_items)] as copyin
}
map [noutput_items] "mapblock_clComplexToArg_kernel4" {
execute  "block_clComplexToArg_kernel5" {
loads [(2*aspen_param_sizeof_FComplex)] as stride(1)
stores [(1*aspen_param_sizeof_float)] as stride(1)
}
call fast_atan2f()
}
execute  "block_clComplexToArg_kernel4__intracommOUT" {
intracomm [(aspen_param_sizeof_float*noutput_items)] as copyout
}
}
```

Programming Systems



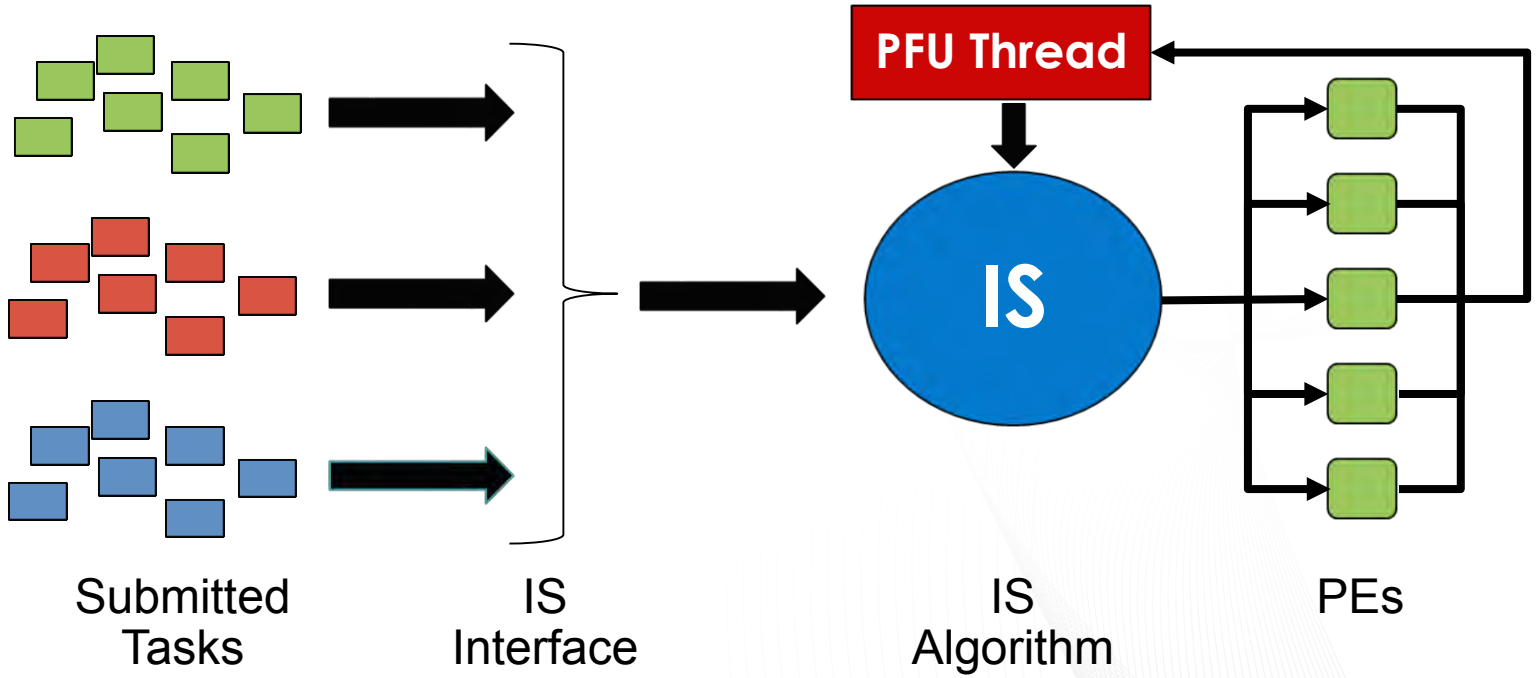
The Minos Computing Library (MCL)

- Framework for programming extremely heterogeneous systems
  - Programming model and programming model runtime
  - Maximize resource utilizations
  - Abstract low-level architecture details from programmers
  - Dynamically schedule work to available resources
- Key programming features:
  - Scheduler dispatches application tasks to available computing resources
  - **Asynchronous** execution of runnable tasks
  - Devices are managed by the scheduler and presented as “Processing Elements” to users
  - **Independent applications** submit tasks without having to synchronize with each other
  - Simplified APIs and programming model (e.g., compared to OpenCL)
- Flexibility:
  - Provides a scheduling framework in which new scheduling algorithm can be plugged in
  - **Multiple scheduling algorithms** co-exist
  - Users don’t need to port code when running on different systems
  - Executing tasks on different PEs doesn’t require user intervention or code modification
  - Resources allocated at the last moment

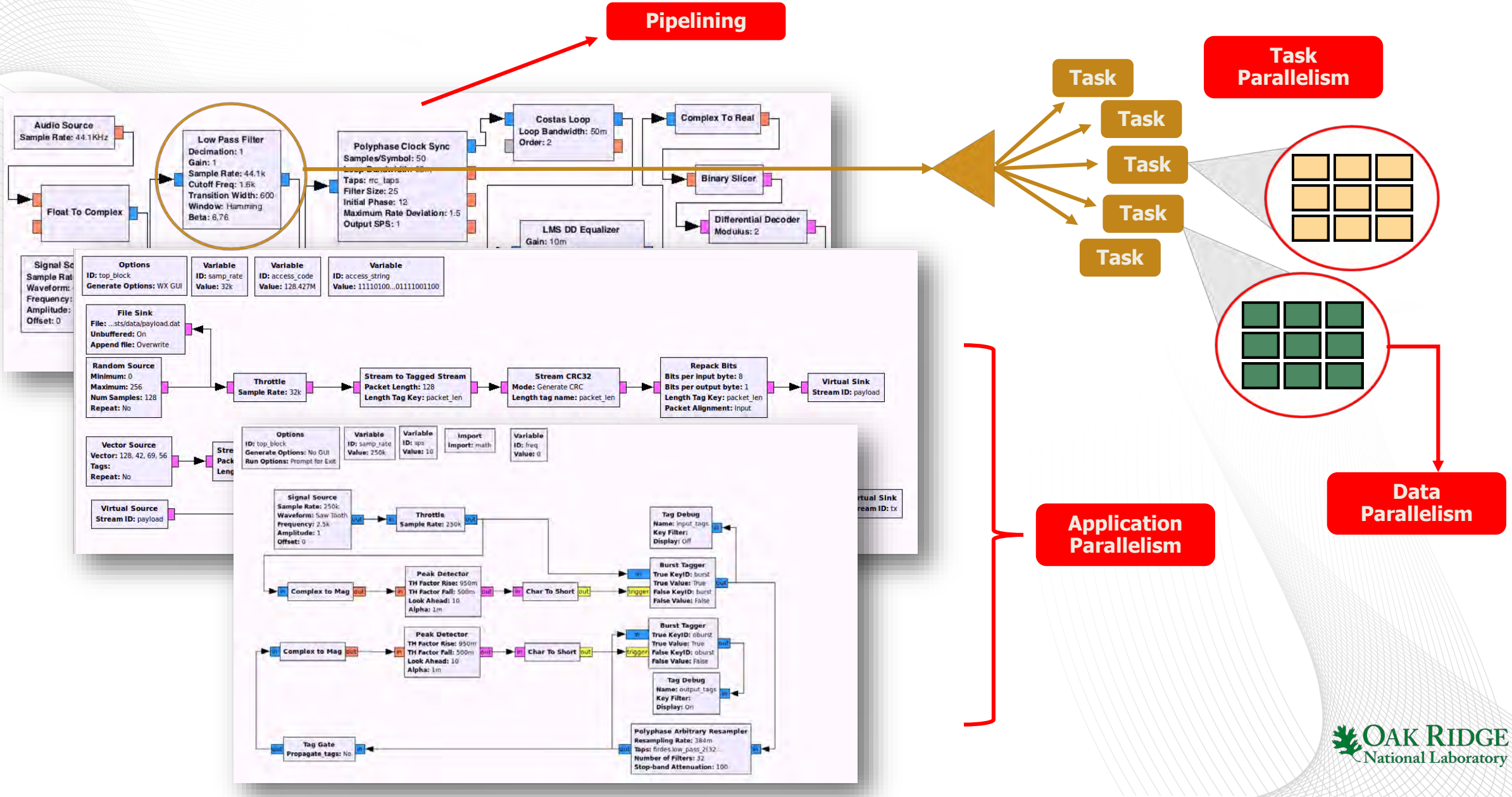


IS Architecture 1/2

- Multiple applications submit tasks
- The IS algorithm dispatches each task to a PE
  - Ontology-based performance expectations
  - Contingent performance and power considerations
  - PFU thread results
- IS collects runtime introspective information
  - System status
  - Task execution characteristics (performance, power, occupancy, etc.)
  - Estimates completion time



Exploiting Parallelism in SDR



MCL Platforms

System	CPU	GPU	Other
NVIDIA Jetson TX1	4-core ARM A57	NVIDIA Maxwell	
NVIDIA Xavier	8-core ARM A57	NVIDIA Volta	2xDLMA
Xilinx ZCU 102	4-core ARM A53	ARM Mali-400 MP2	2-core ARM R5
Apple iMac Pro	16-core Intel Xeon	ATI Radeon Vega	
GPU compute node	2x 20-core Intel Xeon	NVIDIA Pascal	
NVIDIA DGX1 P100	2x 20-core Intel Xeon	8x NVIDIA Pascal	
NVIDIA DGX1 V100	2x 20-core Intel Xeon	4x NVIDIA Volta	Tensor cores
NVIDIA DGX1 V100	2x 20-core Intel Xeon	8x NVIDIA Volta	Tensor cores
IBM Witherspoon	2x 22-core IBM POWER9	6x NVIDIA Volta	Tensor cores

- Currently MCL runs on a variety of architectures and systems
- Time to port to a new system is $\leq$ 1 hour (mostly due to installing OpenCL libraries)
- Same code runs out-of-the-box on new platforms

Conclusions

- Domain Specific Systems on Chip (DSSoC)
  - Developing software and architectures for specific domains
    - SDR
  - Heterogeneity by design
- ORNL project
  - SDR applications
  - Targeting diverse array of heterogeneous hardware including upcoming chips
  - Intelligent programming and runtime system
    - OpenARC
    - MCL
  - Aspen models provides predictions for design, programming, and runtime decisions
- Modeling and simulation challenges
  - Clean slate chip design is too open
    - Need hierarchy of models that allow early exploration through accurate design validation
  - Performance prediction will be necessary to make compilation and scheduling intelligent