Programming the Nallatech Xeon + Multi-FPGA Heterogeneous Platform

Paul Chow, Manuel Saldaña,
Arun Patel, Chris Madill

University of Toronto
ArchES Computing Systems

What are we working with?

We have:

- High-performance multi-core CPUs
- Multiple high-capacity FPGAs; each could contain multiple computing cores
  - Cores can be embedded processors and hardware accelerators, i.e., computing tasks implemented via software or hardware

This is a heterogeneous multi-core platform

Cool box, how do we program it?

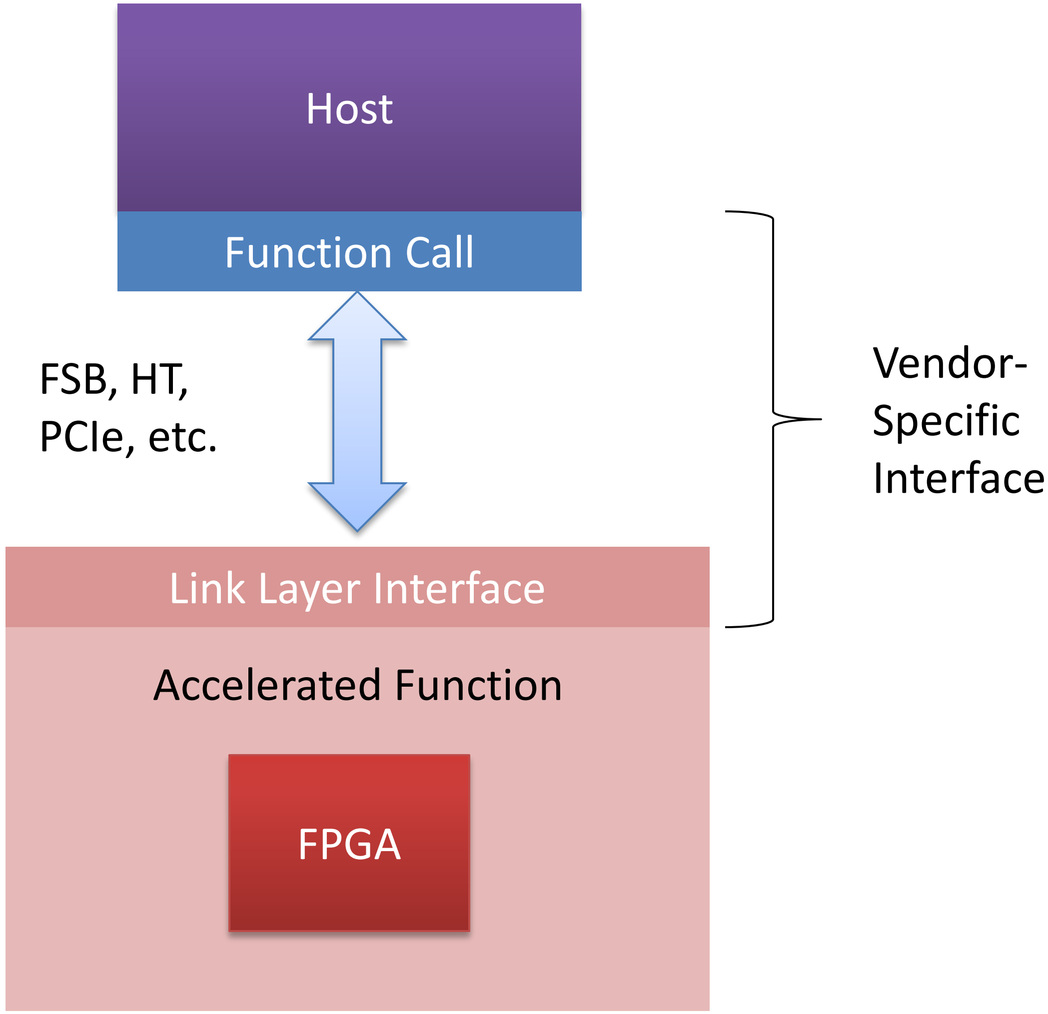
Many programming interfaces to deal with

- Host to host (e.g. sockets)
- Processor to processor in same host (e.g. Pthreads)
- Processor to hardware accelerator (e.g. vendor specific)
- Accelerator to accelerator (e.g. custom design)
- Embedded processors to embedded processors, etc.

Requirements

- Unified programming model
- Usable by application experts, i.e., hide the hardware design as much as possible
- Application portability between platforms
- Application scalability

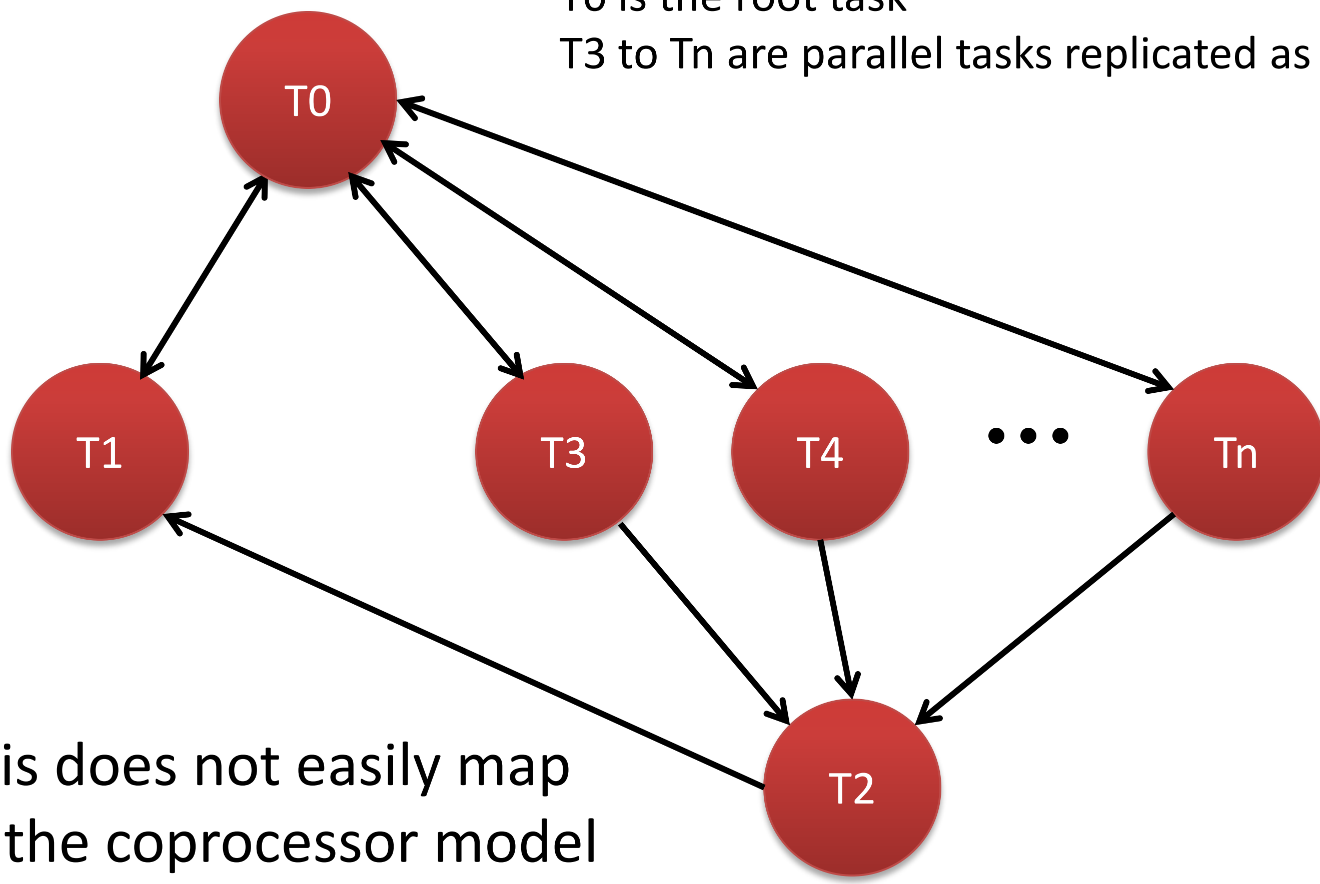
The Simple Coprocessor Model



- Accelerator is a **slave** to software
- Host initiates and controls all interactions
- Does not scale to more accelerators easily or efficiently
- Communication to other Accelerated Functions done via Host

A Representative Parallel Example

T0 is the root task
T3 to Tn are parallel tasks replicated as resources allow



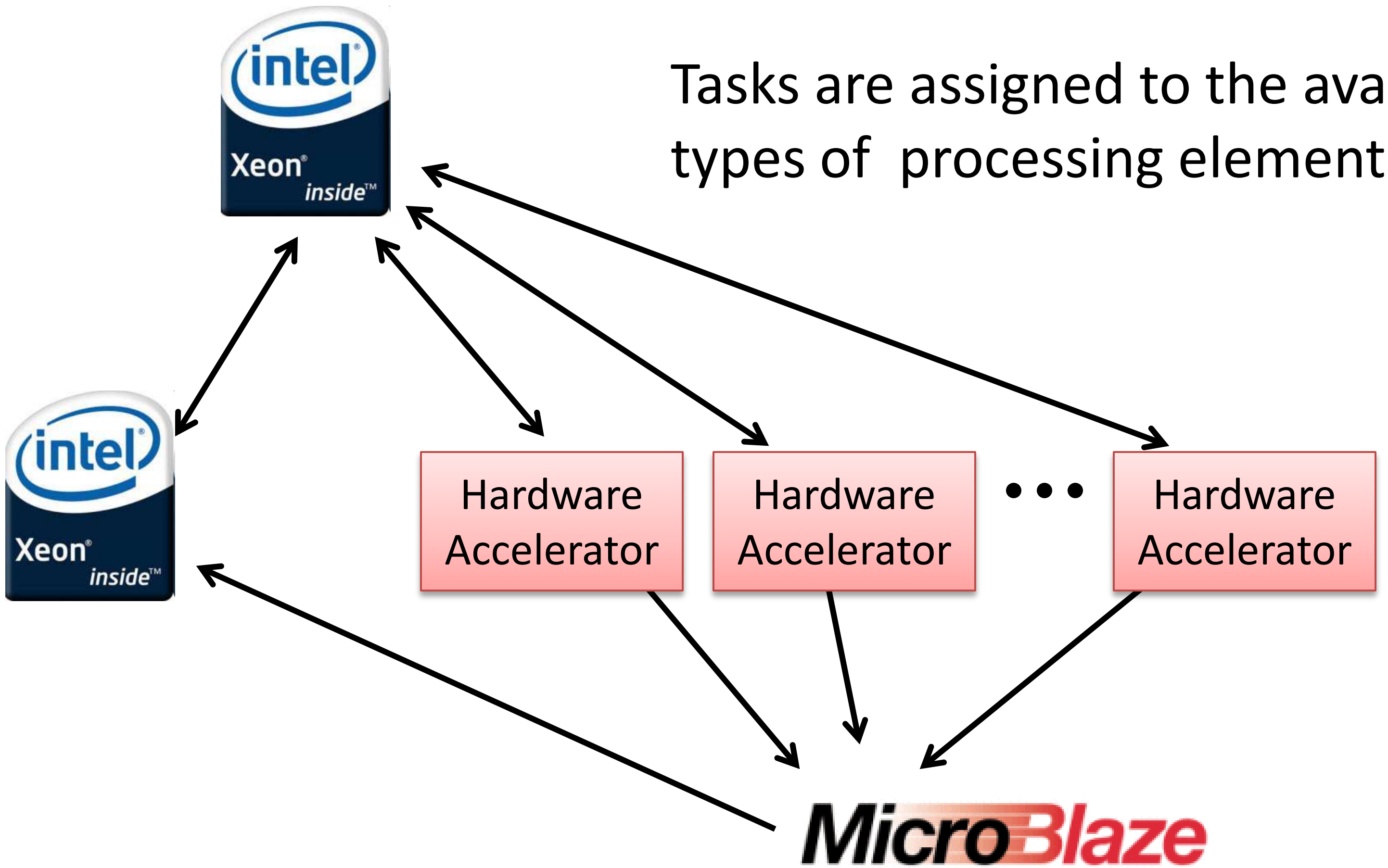
This does not easily map to the coprocessor model

Prototype Stage

- Application Specialist builds the application on a standard cluster using MPI, the Message Passing Interface
- MPI is a common Application Programming Interface used for parallel applications on distributed-memory systems
  - Freely available, supporting tools, large knowledge base
- Working MPI software prototype defines the communication and control protocols used by each task
- Can experiment with task partitioning
- Identify tasks to run in software, accelerate in hardware

Mapping Stage

Tasks are assigned to the available types of processing elements

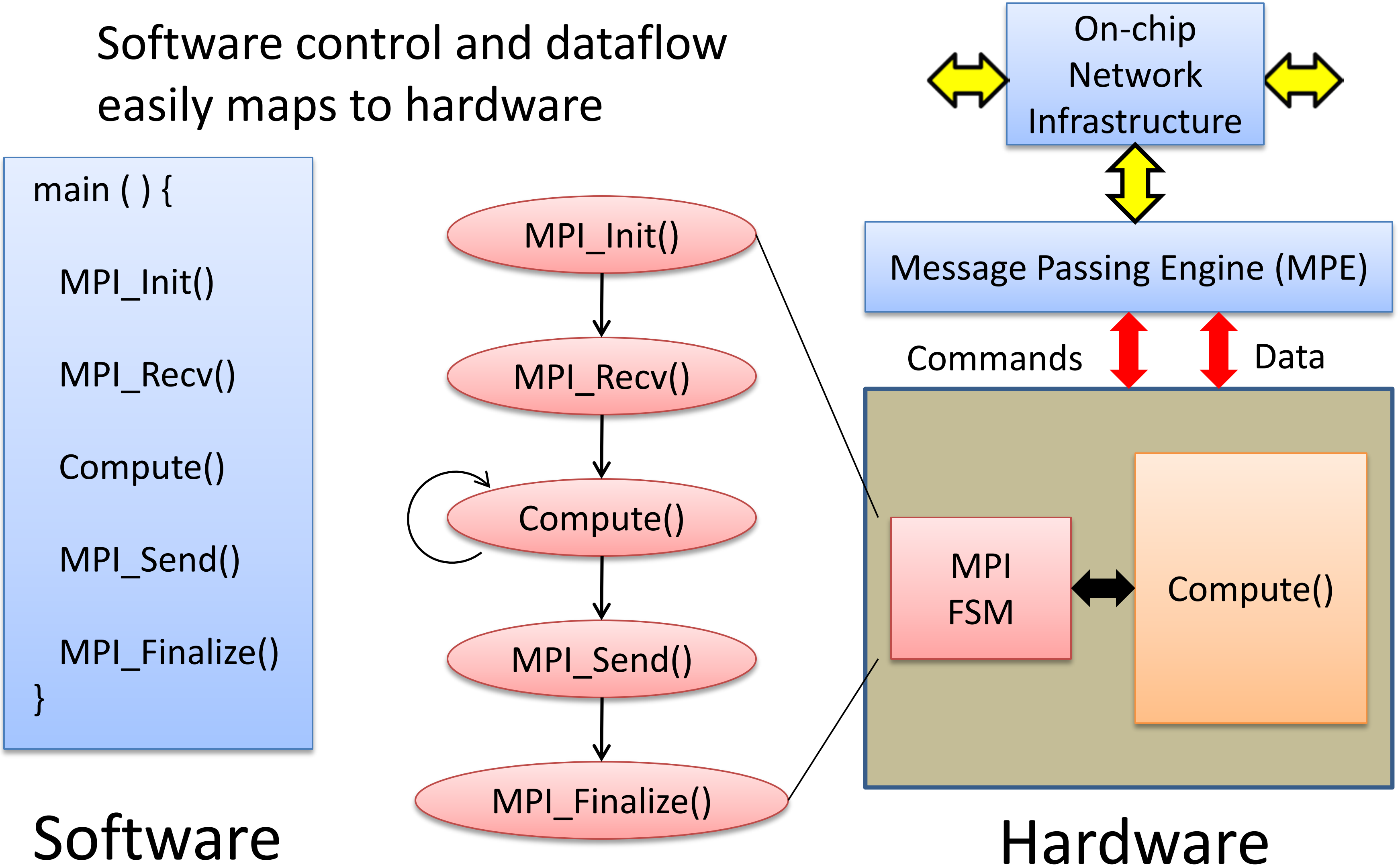


Building the Hardware Accelerators

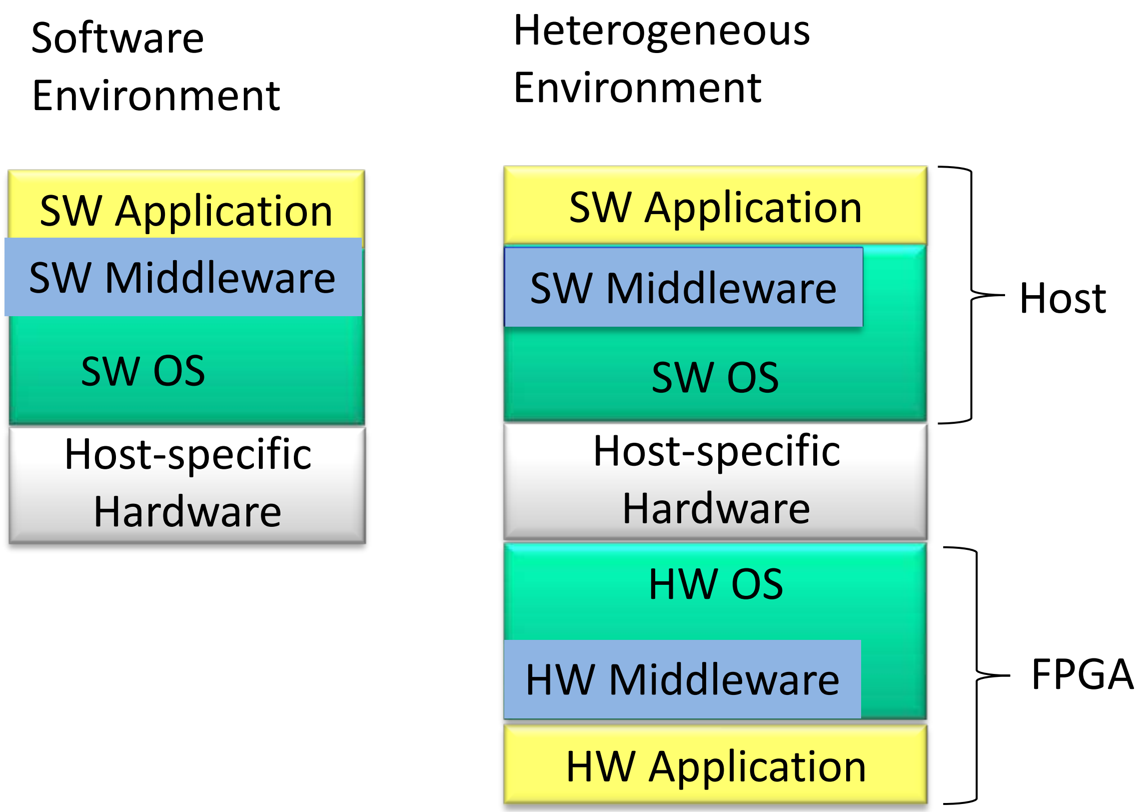
- MPI software version of the task to be converted defines the functionality and interface protocols
- Convert using a hardware designer or a high-level synthesis ("C-to-gates") tool
  - Application expertise not required
- MPI protocols are provided by the MPE (Message Passing Engine) equivalent to the MPI library used by software tasks

Accelerator Architecture

Software control and dataflow easily maps to hardware

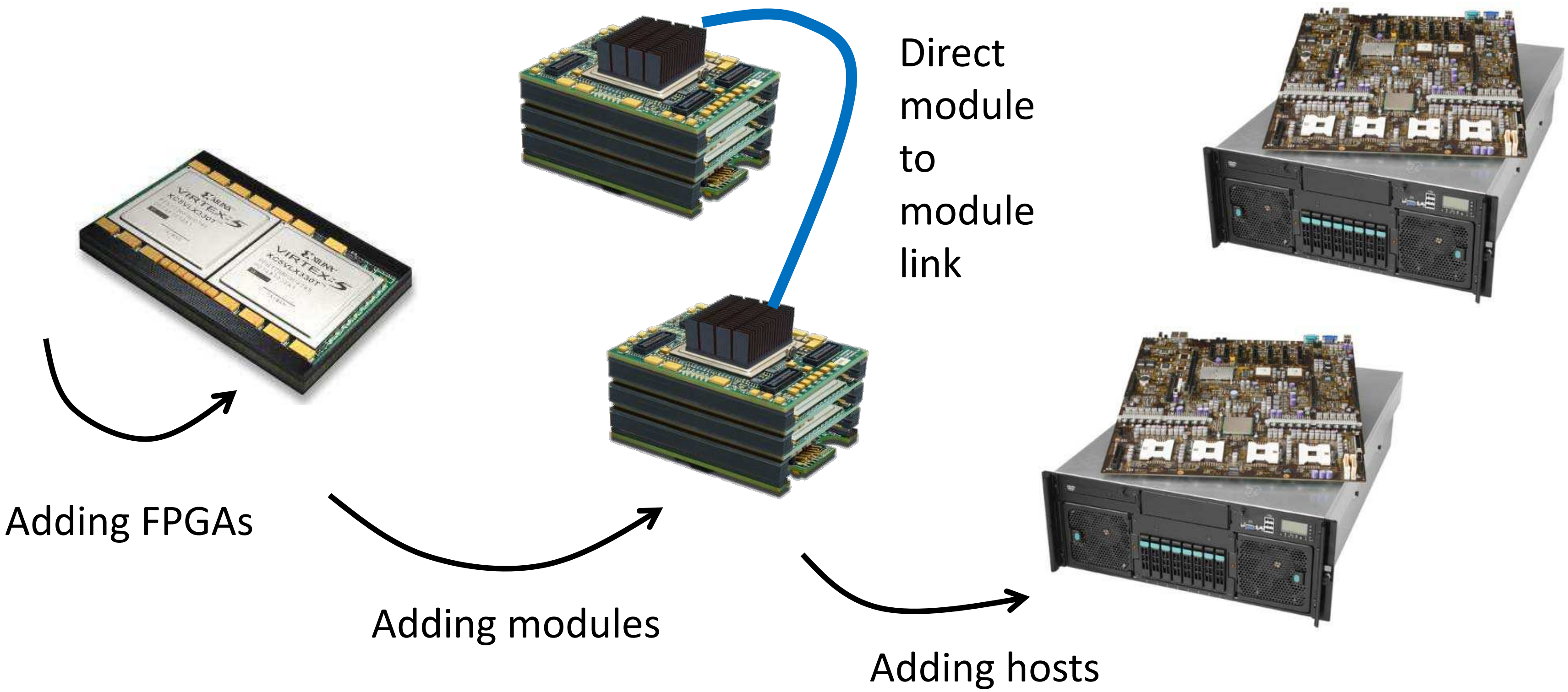


Achieving Portability with ArchES MPI



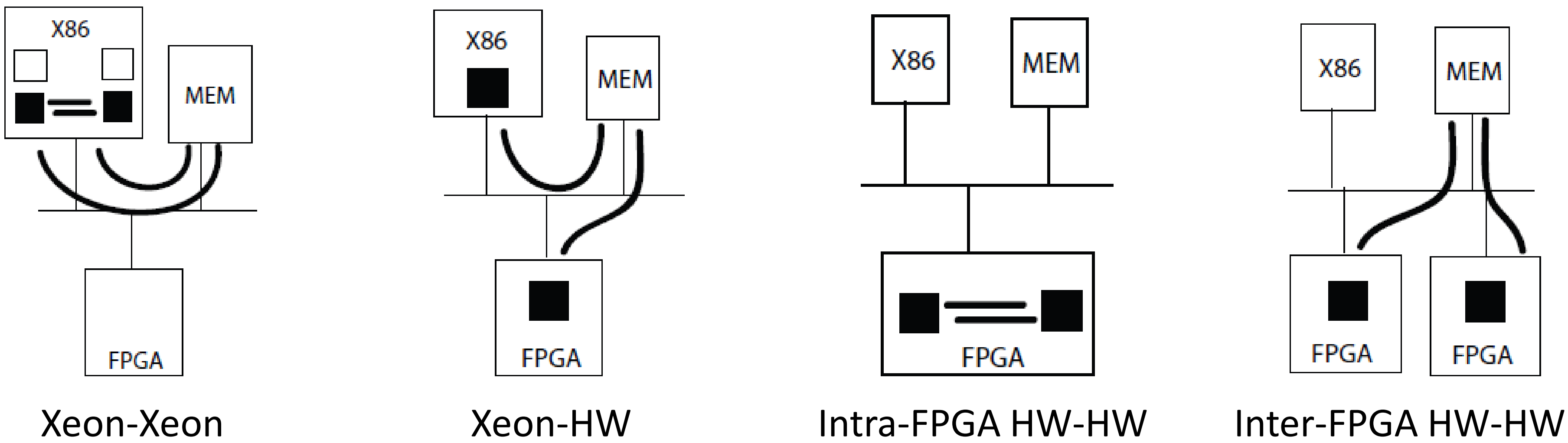
- Portability is achieved by using a Middleware abstraction layer. MPI natively provides software portability
- ArchES MPI provides a Hardware Middleware to enable hardware portability. The MPE provides the portable hardware interface to be used by a hardware accelerator

Achieving Scalability with ArchES MPI



- MPI naturally enables scalability
- Making use of additional processing capability can be as easy as changing a configuration file
- Scaling at the different levels (FPGAs, modules, hosts) is transparent to the application

Configurations for Performance Testing



Send round-trip messages between two MPI tasks (black squares)
X86 has Xeon cores using software MPI, FPGA has hardware engines (HW) using the MPE

Δt = round_trip_time/(2*num_samples)
Latency = Δt for a small message size
BW = message_size/Δt

Measurements here are done using only FSB-Base modules. We can do this also with the FSB-Compute and FSB-Expansion Modules by moving the location of the HW

Preliminary Performance Numbers

On-chip network using 32-bit channels and clocked at 133 MHz
MPI using Rendezvous Protocol

	Xeon-Xeon	Xeon-HW	HW-HW (intra-FPGA)	HW-HW (inter-FPGA)
Latency [μs] (64-byte transfer)	1.9	2.78	0.39	3.5
Bandwidth [MB/s]	1000	410	531	400

- Xilinx driver performance numbers
  - Latency = 0.5 μs (64 byte transfer)
  - Bandwidth = 2 GB/s

- MPI Ready Protocol achieves about 1/3 of the Rendezvous latency. For Xeon-HW it is 1μs (only 2X slower than Xilinx driver transfer latency)

- 128-bit on-chip channels will quadruple the HW bandwidth (to approx. 2GB/s) and also reduce latency
  - ArchES is implementing other performance enhancements

Conclusions

- Application architecture is abstracted from the implementation by the MPI layer
    - MPI provides a unified programming interface
- Hardware implementation can be done without requiring application expertise so that the Application specialist can focus on application design
- MPI naturally supports scalability
- MPI naturally provides portability
- ArchES MPI enables high-performance, multi-core applications for the Nallatech Xeon + multi-FPGA heterogeneous computing platform

Research Support

NSERC CRSNG

CIHR IRSC

CMC MICROSYSTEMS SOCRN

XILINX®

SickKids®

CONACYT

AMIRIX
our guts...your glory™

Thank you

Professor Paul Chow
University of Toronto
pc@eecg.toronto.edu

www.archescomputing.com