MPI as a Programming Model and Development Environment for Heterogeneous Computing (with FPGAs)

Paul Chow

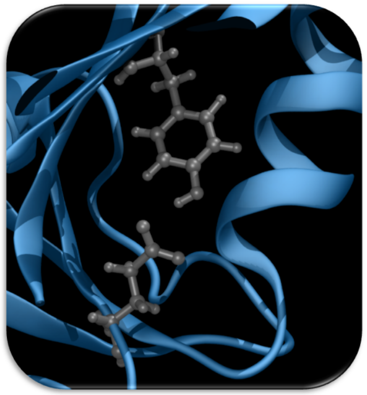
University of Toronto
Department of Electrical and Computer Engineering

ArchES Computing Systems

June 10, 2012

Let's Build a Big Machine

- FPGAs seen to provide significant acceleration in "one's and two's"
- What about thousands?
- Big application? – Molecular Dynamics



$$U_t = \sum_i \begin{cases} k_i\left[1+\cos\left(n_i\varphi_i-\gamma_i\right)\right], n_i \neq 0 \\ k_i\left(0_i-\gamma_i\right)^2, n=0 \end{cases}$$

$$U_a = \sum_i k_i(\theta_i - \theta_{0i})^2$$

$$U_b = \sum_i k_i(r_i - r_{0i})^2$$

O(n)

O(n²)

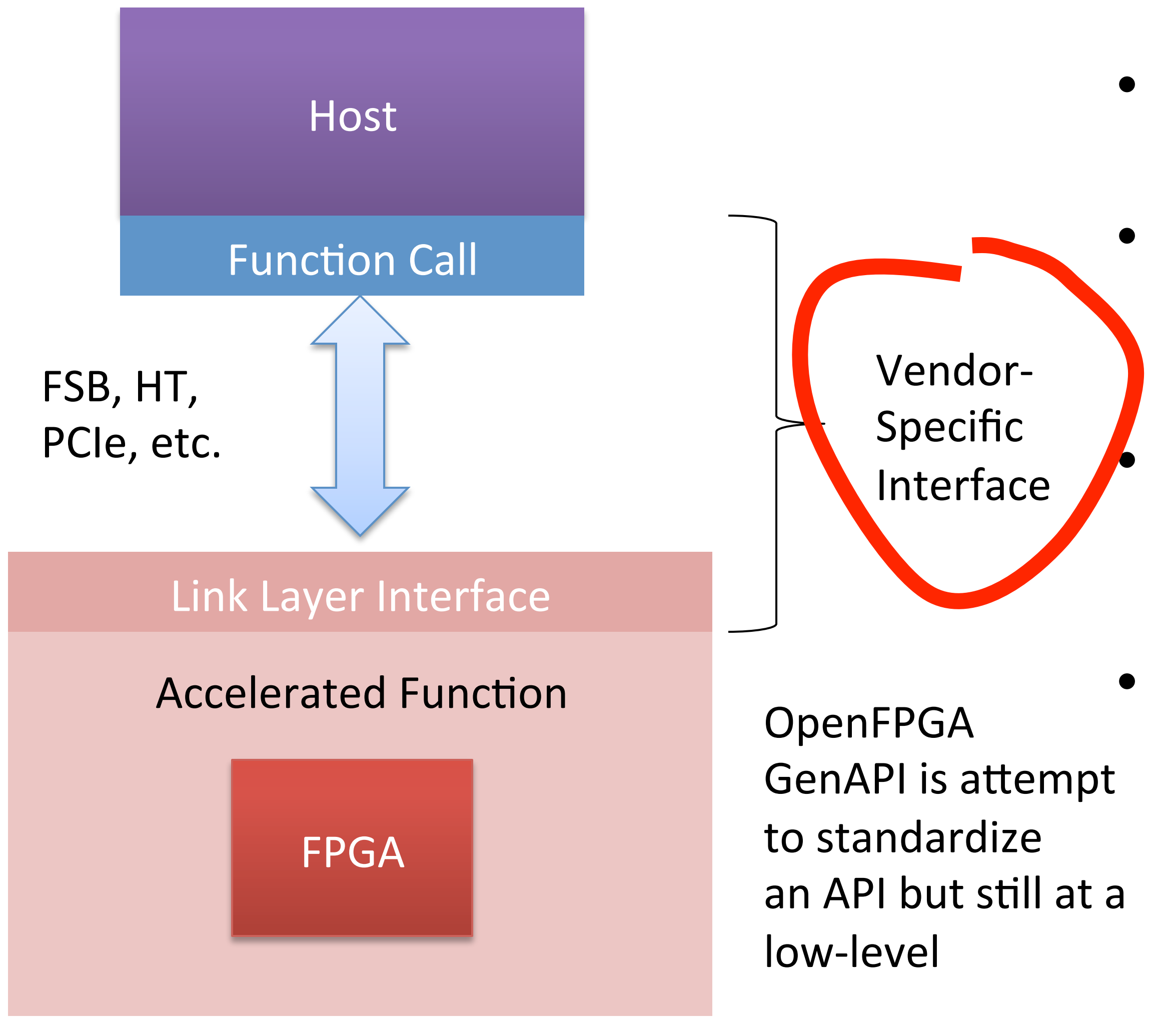
$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

$$U = \frac{1}{2}\sum_{\vec{n}}{}' \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{q_i q_j}{\left|\vec{r}_{ij}+\vec{n}\right|}$$

How do you think about such a system?

- Not just a hardware circuit with inputs and outputs
- It's a purpose-built computing machine
- Desire programmability, scalability, reuse, maintainability
- Initially all FPGAs, now includes x86 systems
- Needs a programming model – BTW, hardware designers don't think this way…

The Typical Accelerator Model



OpenFPGA GenAPI is attempt to standardize an API but still at a low-level

- 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

Many interfacing and communication issues before even thinking about the application. Not to forget the hardware design!

Requirements for Programming Model

- Unified – same API for all communication
- Usable by application experts, i.e., hide the heterogeneity – it's just a collection of processors
- Application portability between platforms
- Application scalability

Leverage existing software programming models and adapt to heterogeneous environment

Which Programming Model?

- Hardware – chunks of logic with attached local memory
- Sounds like distributed memory

Message Passing $\rightarrow$ MPI

Using MPI

- 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
  - Lots to leverage from

Version Guide

TMD-MPI ⟷ ArchES-MPI

Original research project

Commercialized

Will see both names in publications.
Treat as equivalent here.

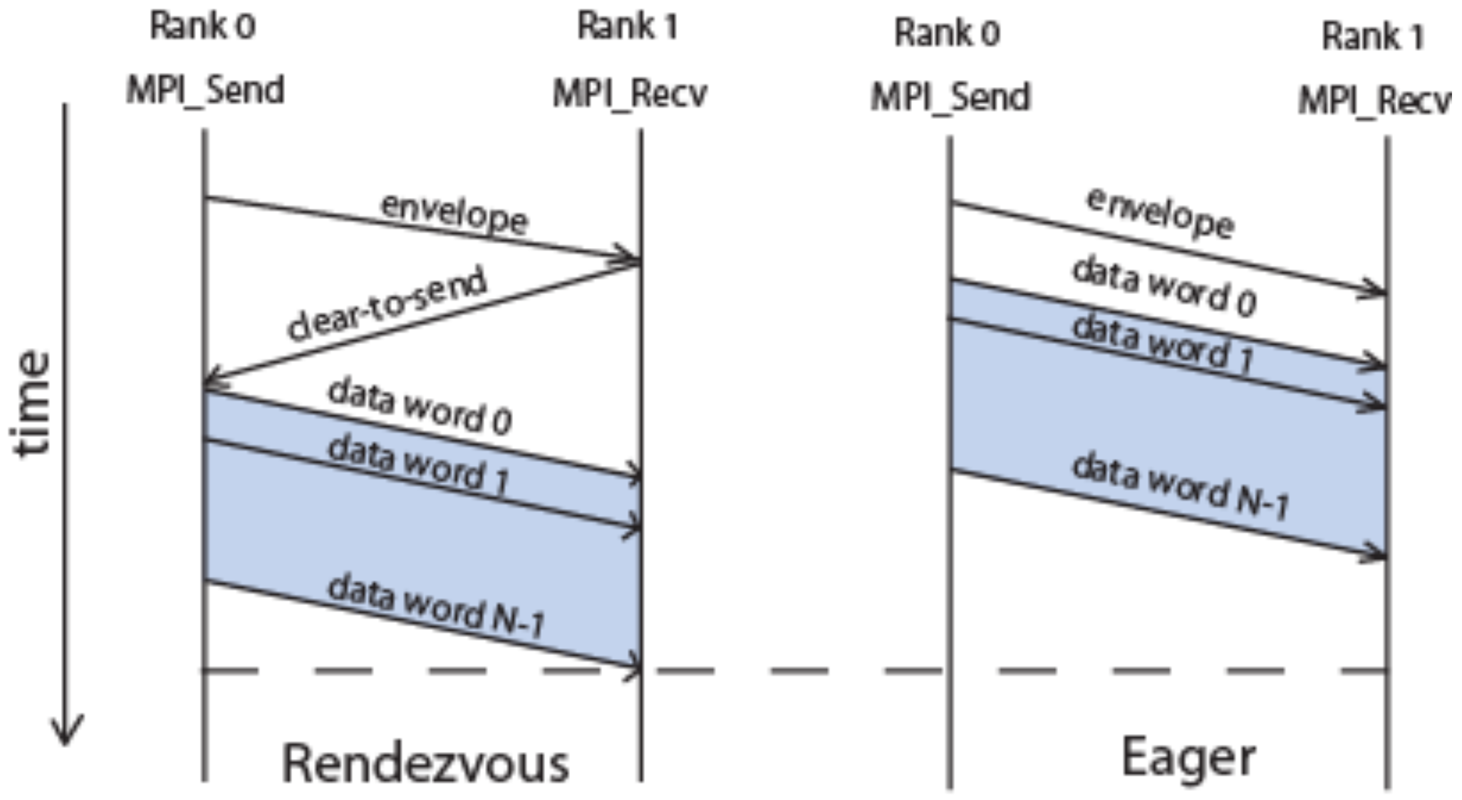
An “Embedded MPI”

Saldaña, et al., ACM TRETS, Vol. 3, No. 4, November 2010

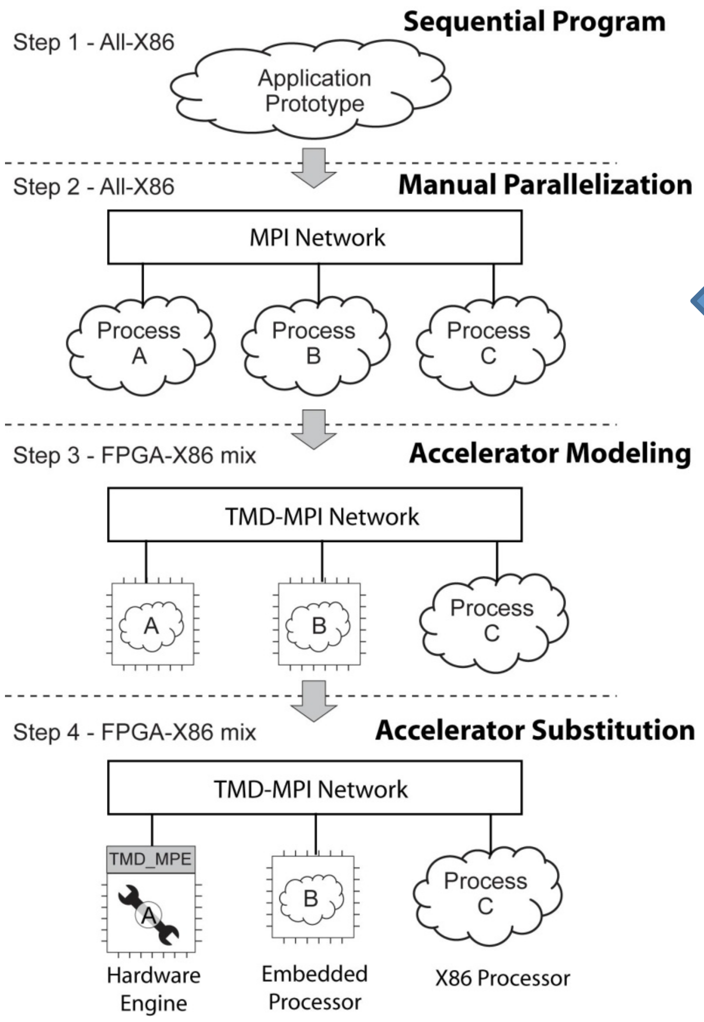
- Lightweight subset of the MPI standard
- Tailored to a particular application
- No operating system required
- Small memory footprint ~8.7KB
- Simple protocol
- Used as a model for implementing a message-passing engine in hardware – the MPE
- Abstraction isolates software from hardware changes providing portability

Protocols

TMD-MPI communication protocols



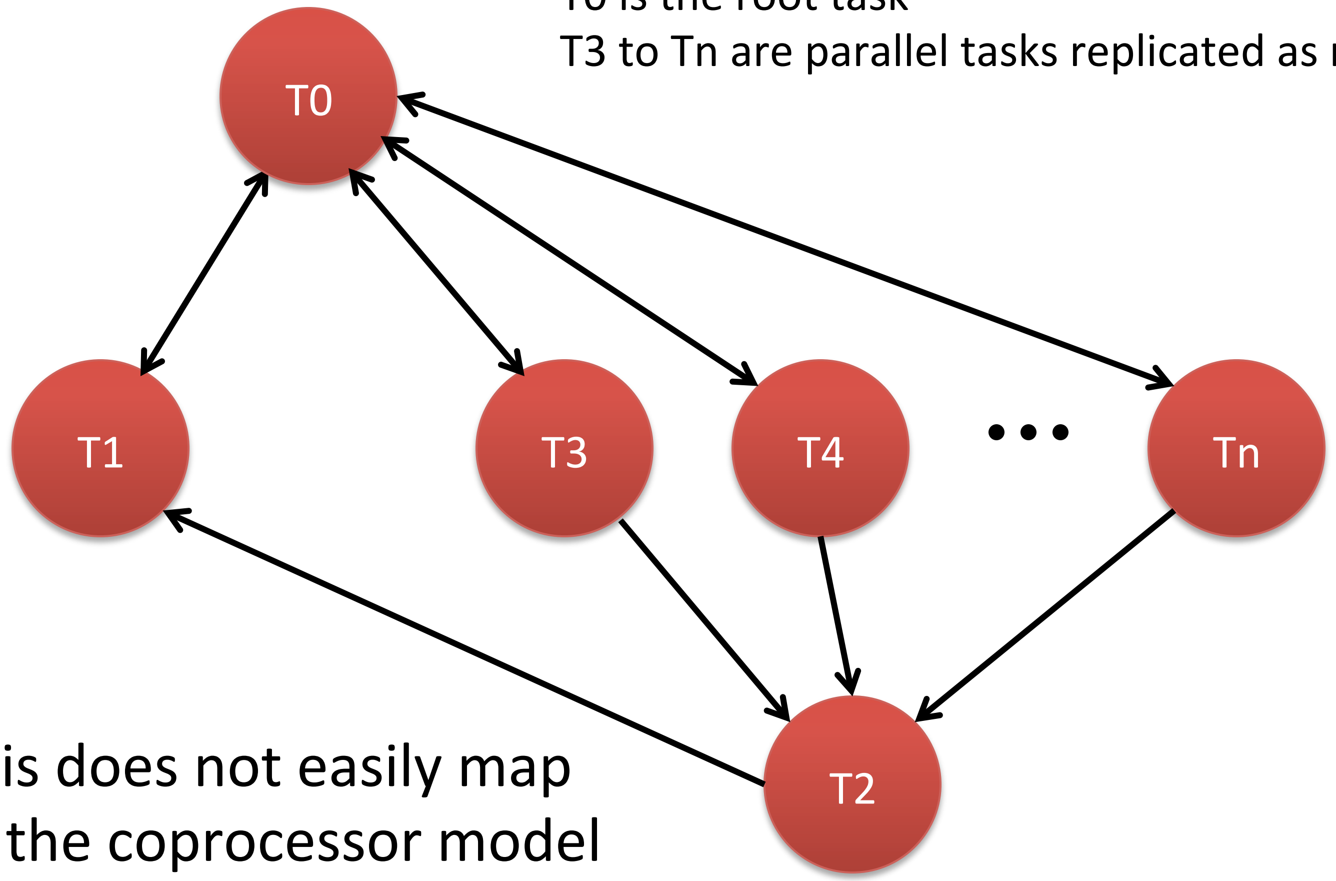
The Flow



Also a system simulation

A Parallel Example

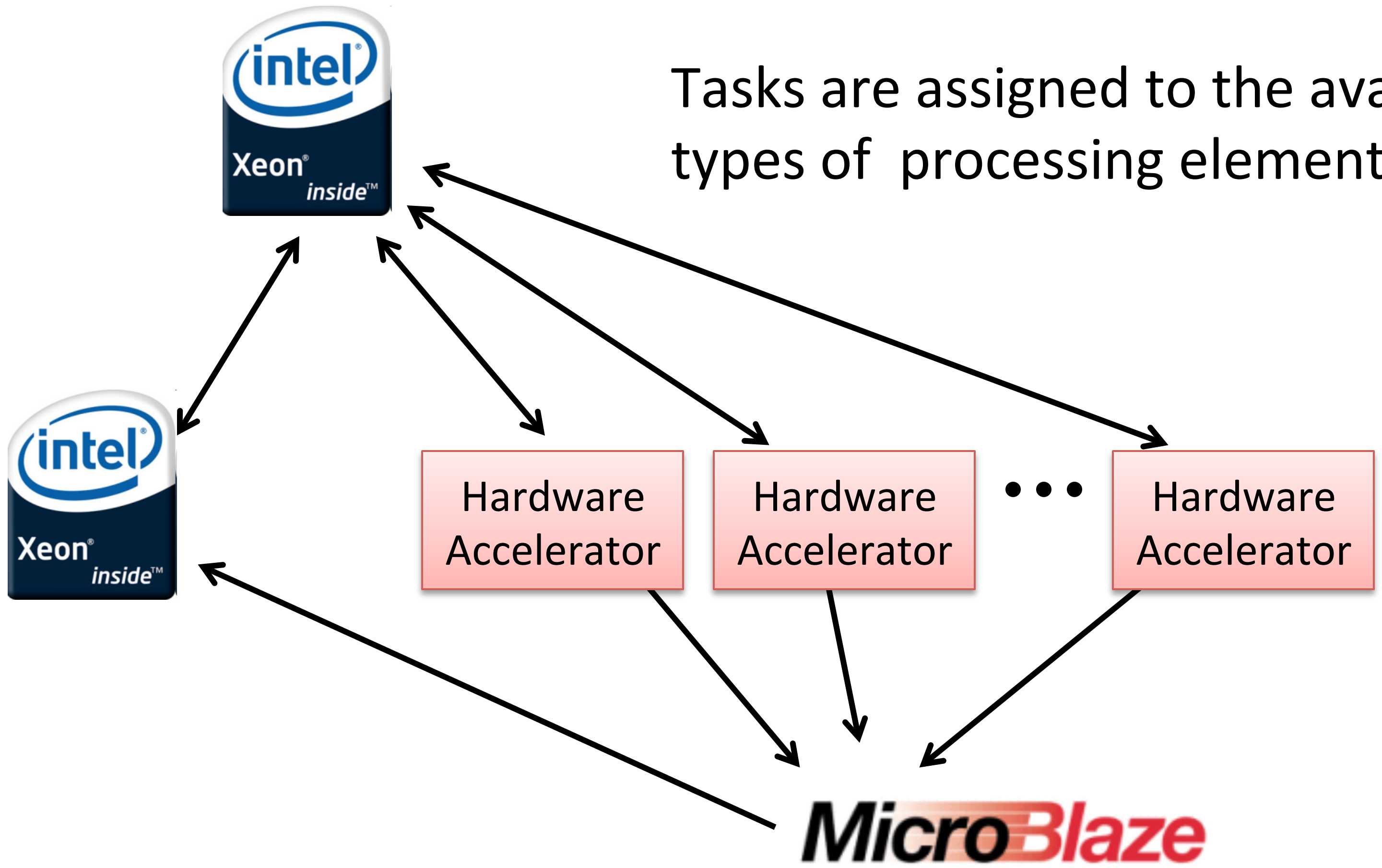
T0 is the root task
T3 to Tn are parallel tasks replicated as resources allow



This does not easily map to the coprocessor model

Mapping Stage

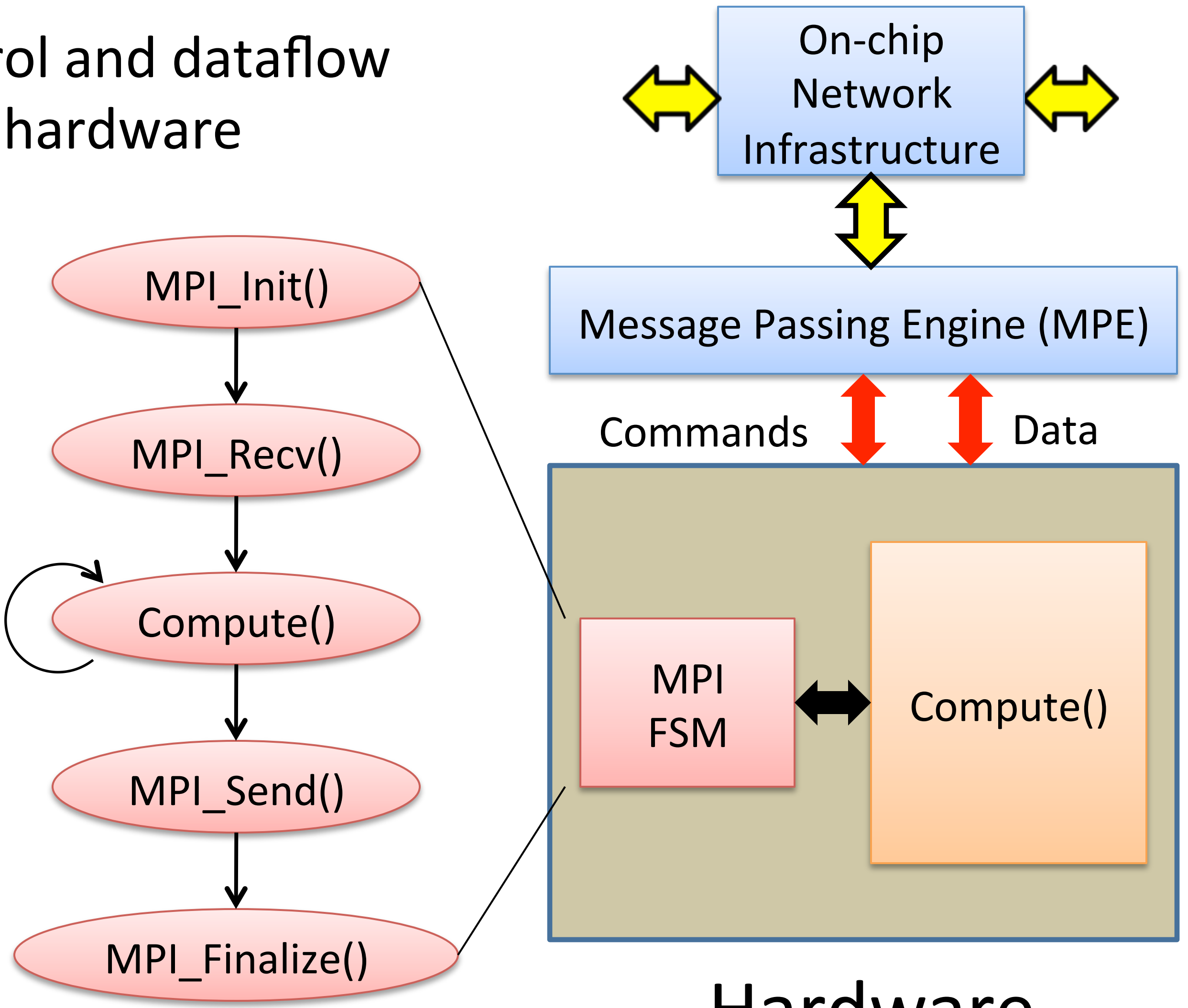
Tasks are assigned to the available types of processing elements



Accelerator Architecture

Software control and dataflow easily maps to hardware

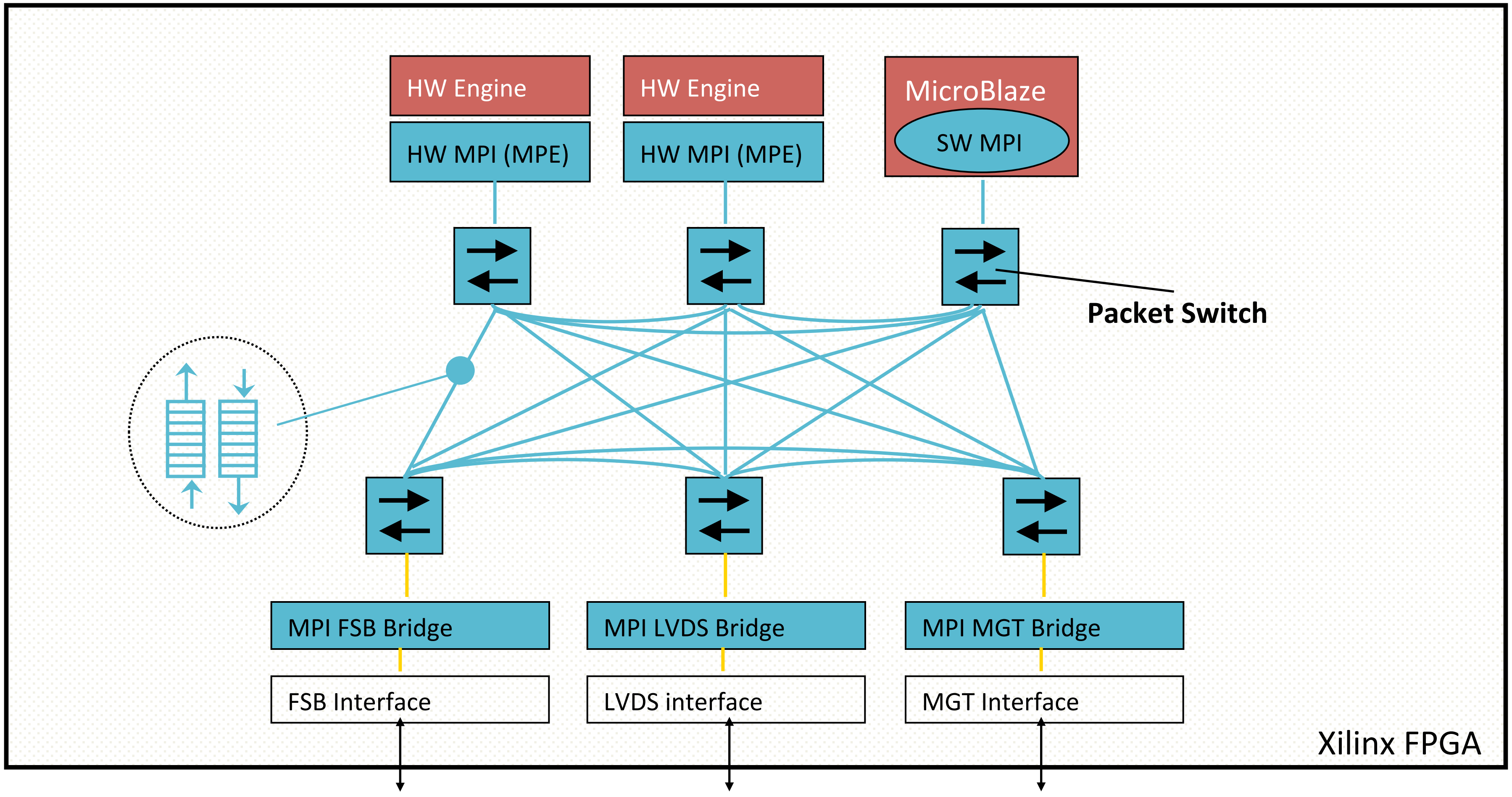
```
main ( ) {
   MPI_Init()
   MPI_Recv()
   Compute()
   MPI_Send()
   MPI_Finalize()
}
```



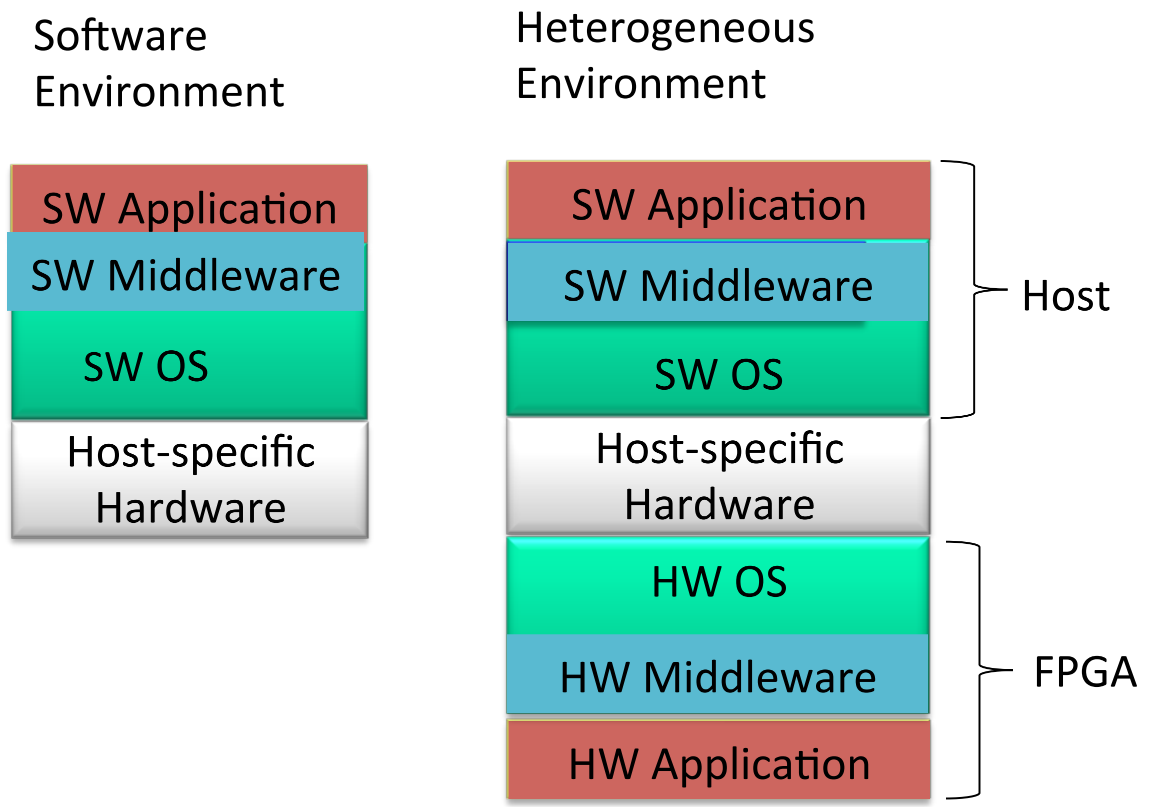
Software

Hardware

Communication Middleware

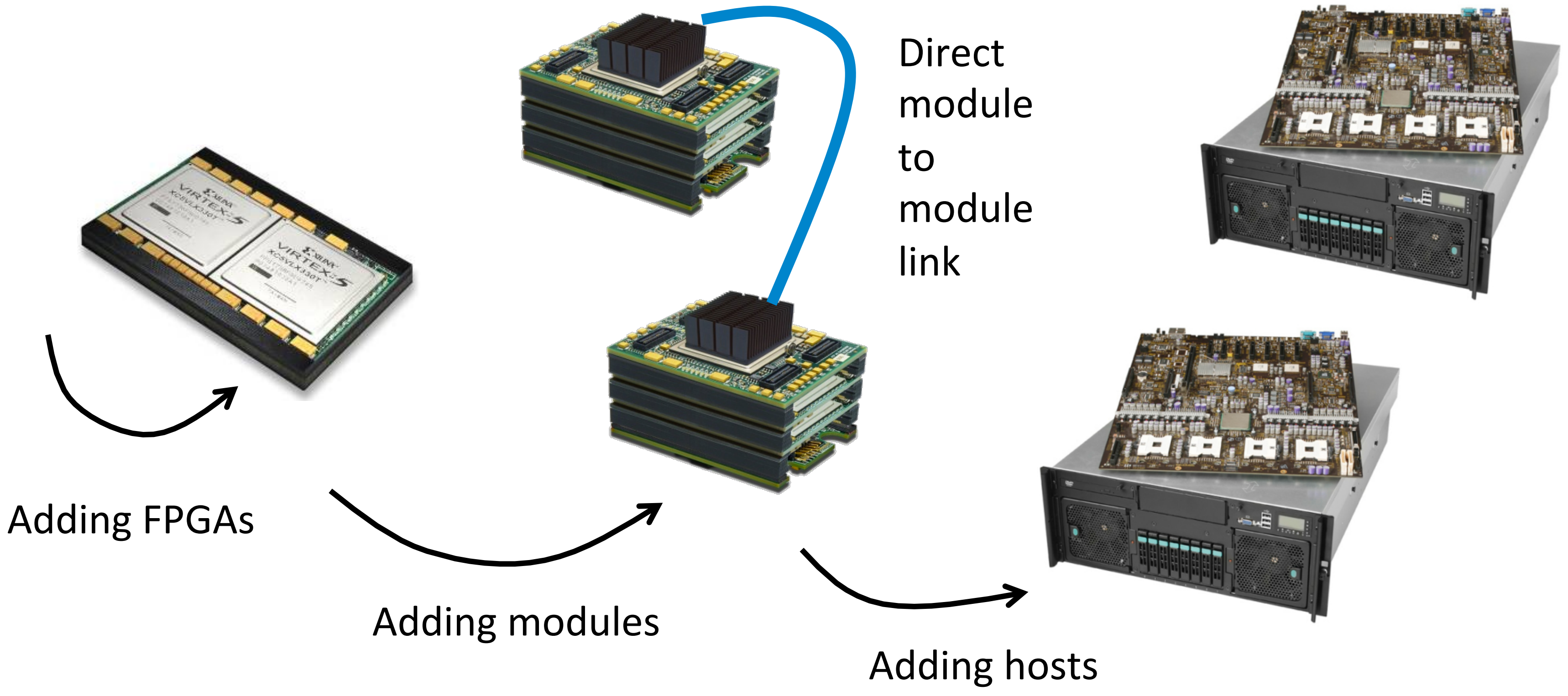


Achieving Portability



- 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



- 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

BUILDING A LARGE HPC APPLICATION

Molecular Dynamics

- Simulate motion of molecules at atomic level
- Highly compute-intensive
- Understand protein folding
- Computer-aided drug design

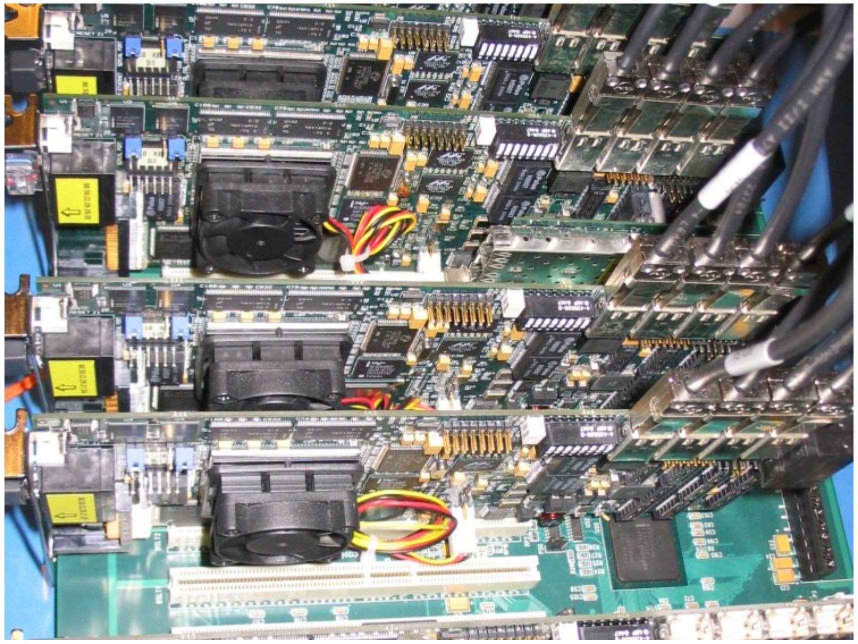
The TMD Machine

- The Toronto Molecular Dynamics Machine
- Use multi-FPGA system to accelerate MD
- Principal algorithm developer: Chris Madill, Ph.D. candidate (now done!) in Biochemistry
  - Writes C++ using MPI, **not** Verilog/VHDL
- Have used three platforms – portability

Platform Evolution

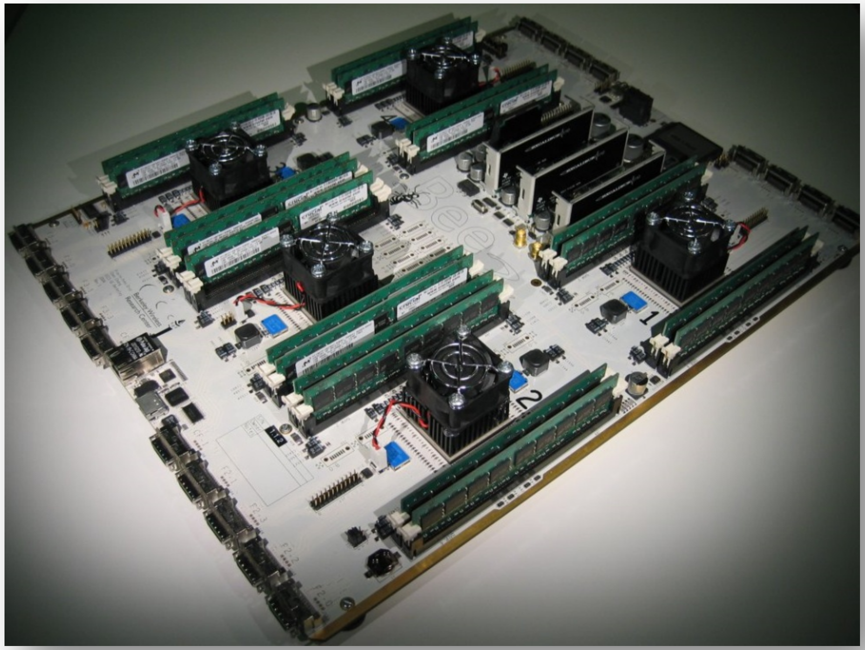
FPGA portability and design abstraction facilitated ongoing migration.

Network of Five V2Pro PCI Cards (2006)



- First to integrate hardware acceleration
- Simple LJ fluids only

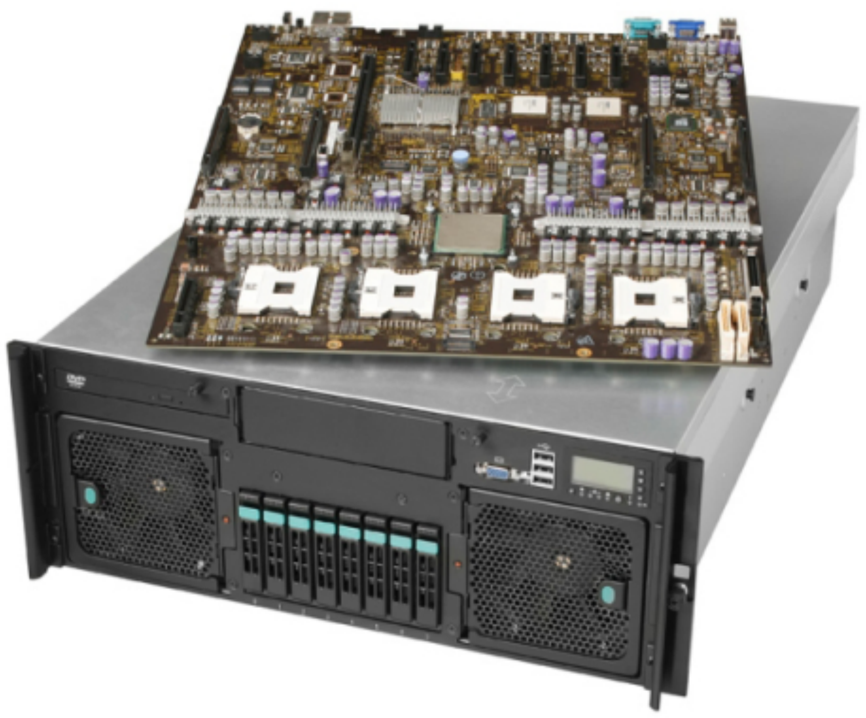
Network of BEE2 Multi-FPGA Boards (2007)



- Added electrostatic terms
- Added bonded terms

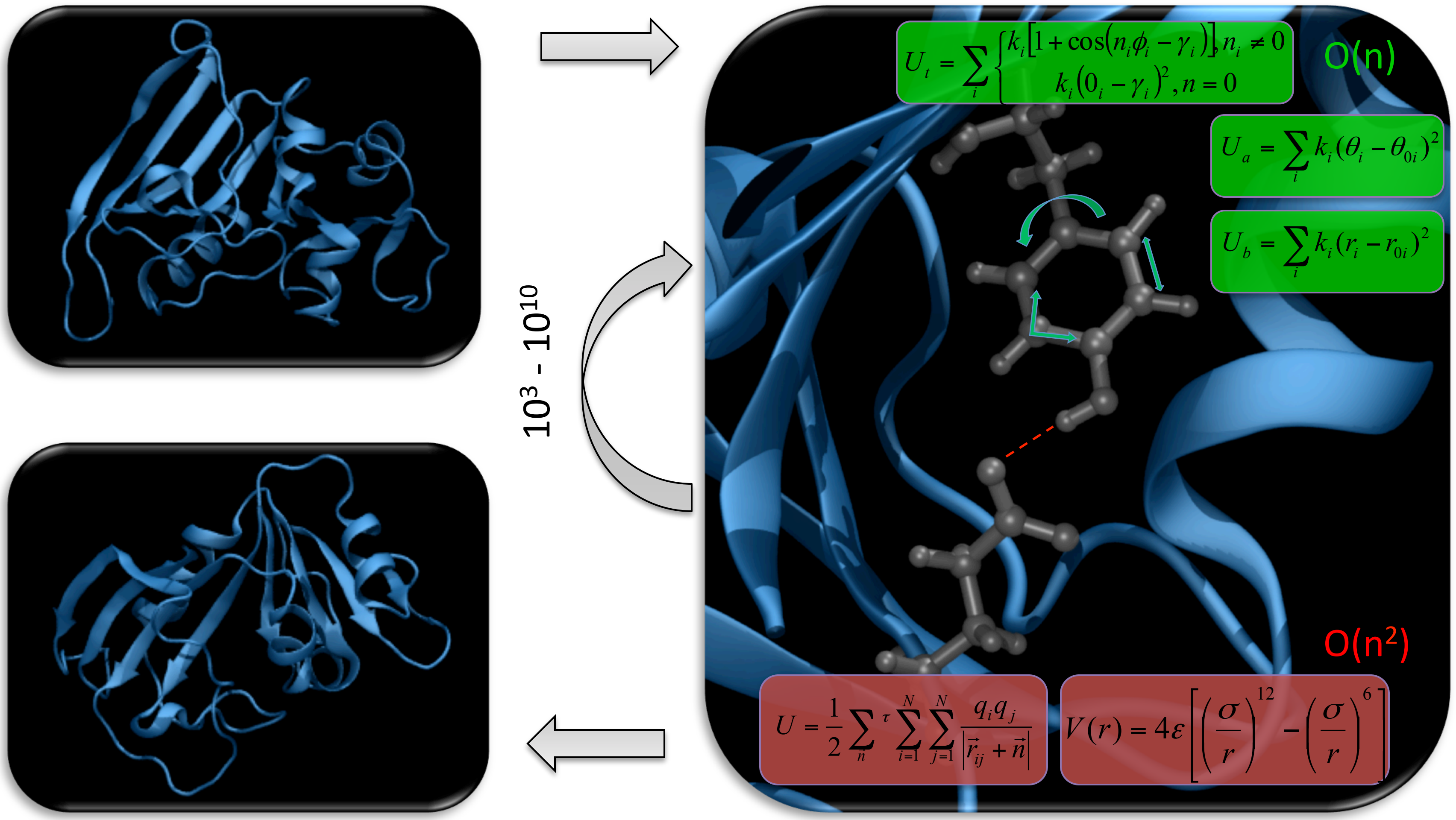
2010 – Xilinx/Nallatech ACP

Stack of 5 large Virtex-5 FPGAs + 1 FPGA for FSB PHY interface



Quad socket Xeon Server

Origin of Computational Complexity



$$U_t = \sum_i \begin{cases} k_i\left[1+\cos\left(n_i\phi_i-\gamma_i\right)\right], n_i \neq 0 \\ k_i\left(0_i-\gamma_i\right)^2, n=0 \end{cases}$$

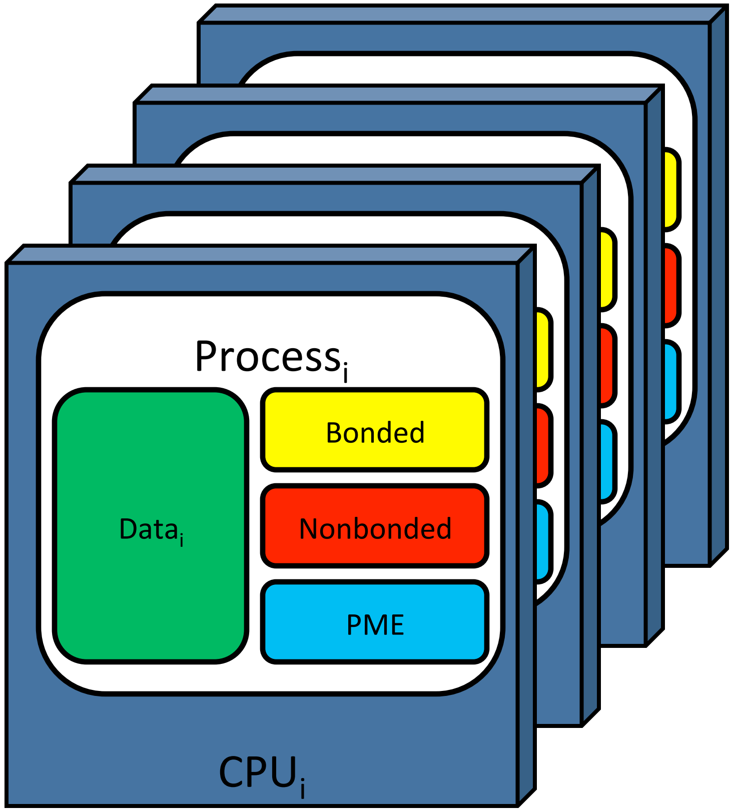
$$U_a = \sum_i k_i(\theta_i-\theta_{0i})^2$$

$$U_b = \sum_i k_i(r_i-r_{0i})^2$$

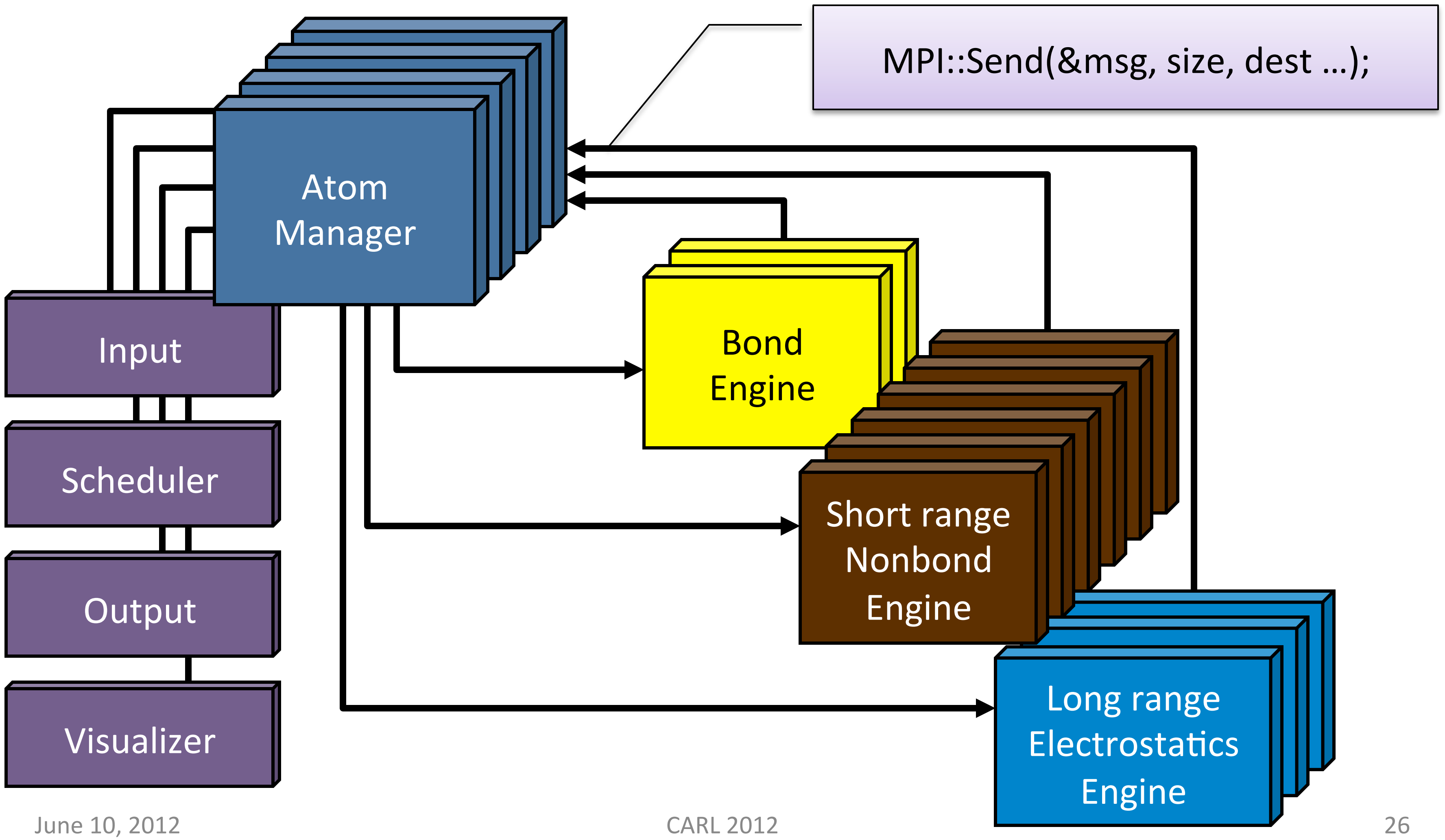
$$U = \frac{1}{2}\sum_n{}^{\tau}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{\left|\vec{r}_{ij}+\vec{n}\right|}$$

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^{6}\right]$$

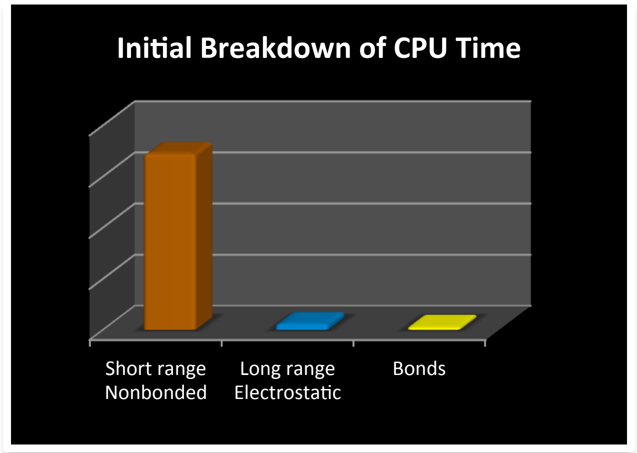
Typical MD Simulator



TMD Machine Architecture


MPI::Send(&msg, size, dest …);
Atom Manager
Input
Scheduler
Output
Visualizer
Bond Engine
Short range Nonbond Engine
Long range Electrostatics Engine

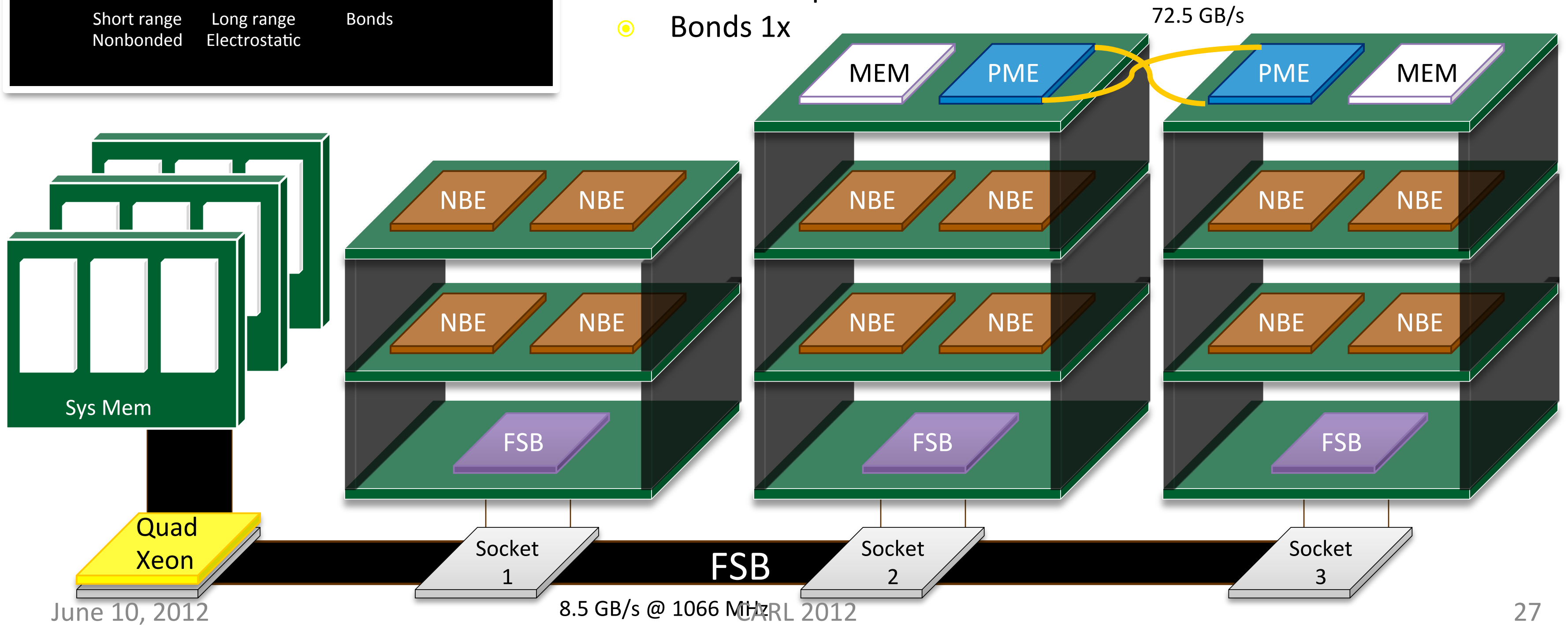
Target Platform for MD



- 12 short range nonbond FPGAs
- 2-3 pipelines/NBE FPGA; Each runs 15-30x CPU
- NBE 360-1080x

- 2 PME FPGAs with fast memory and fibre optic interconnects
- PME 420x

- Bonds on quad-core Xeon server
- Bonds 1x

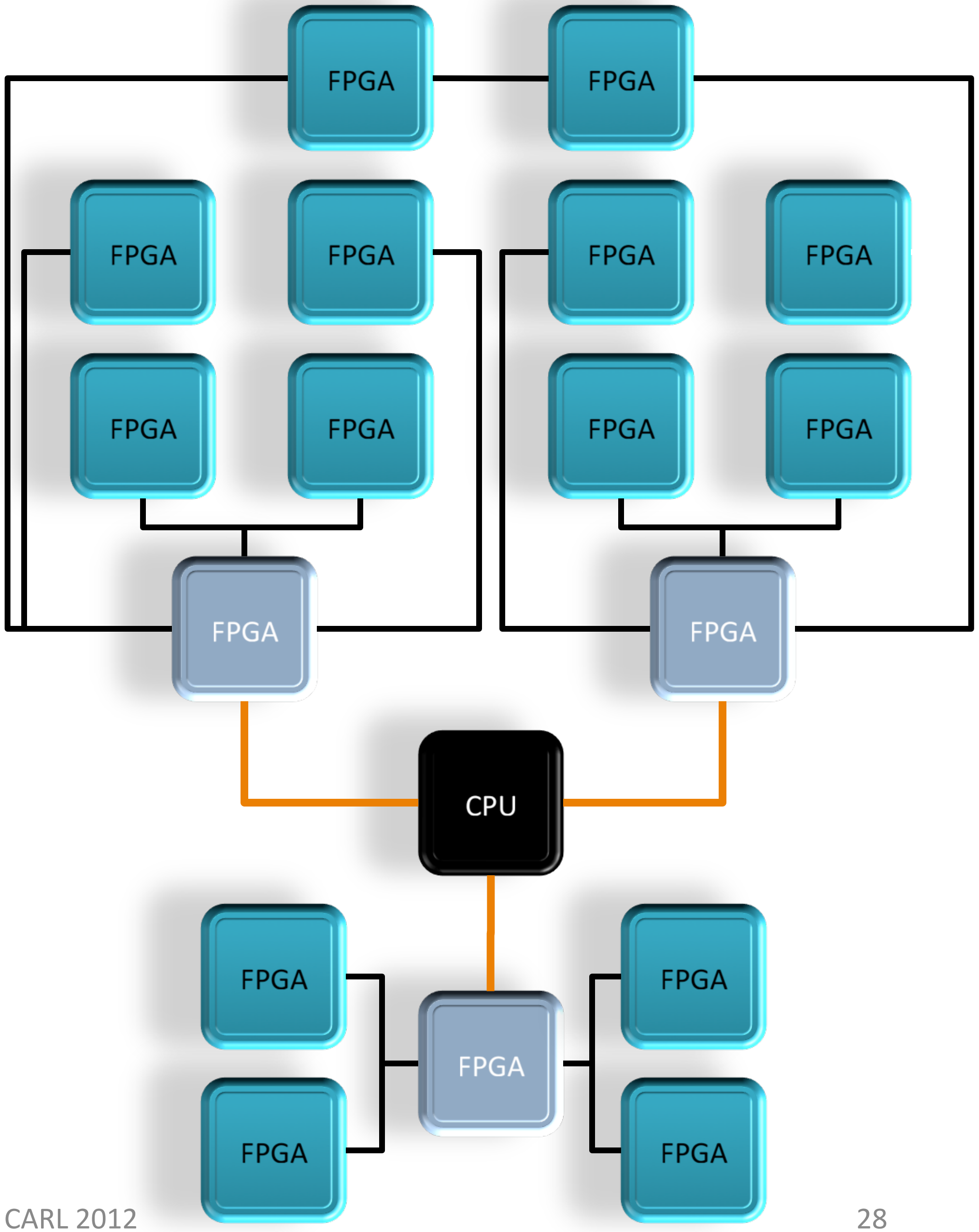


Performance Modeling

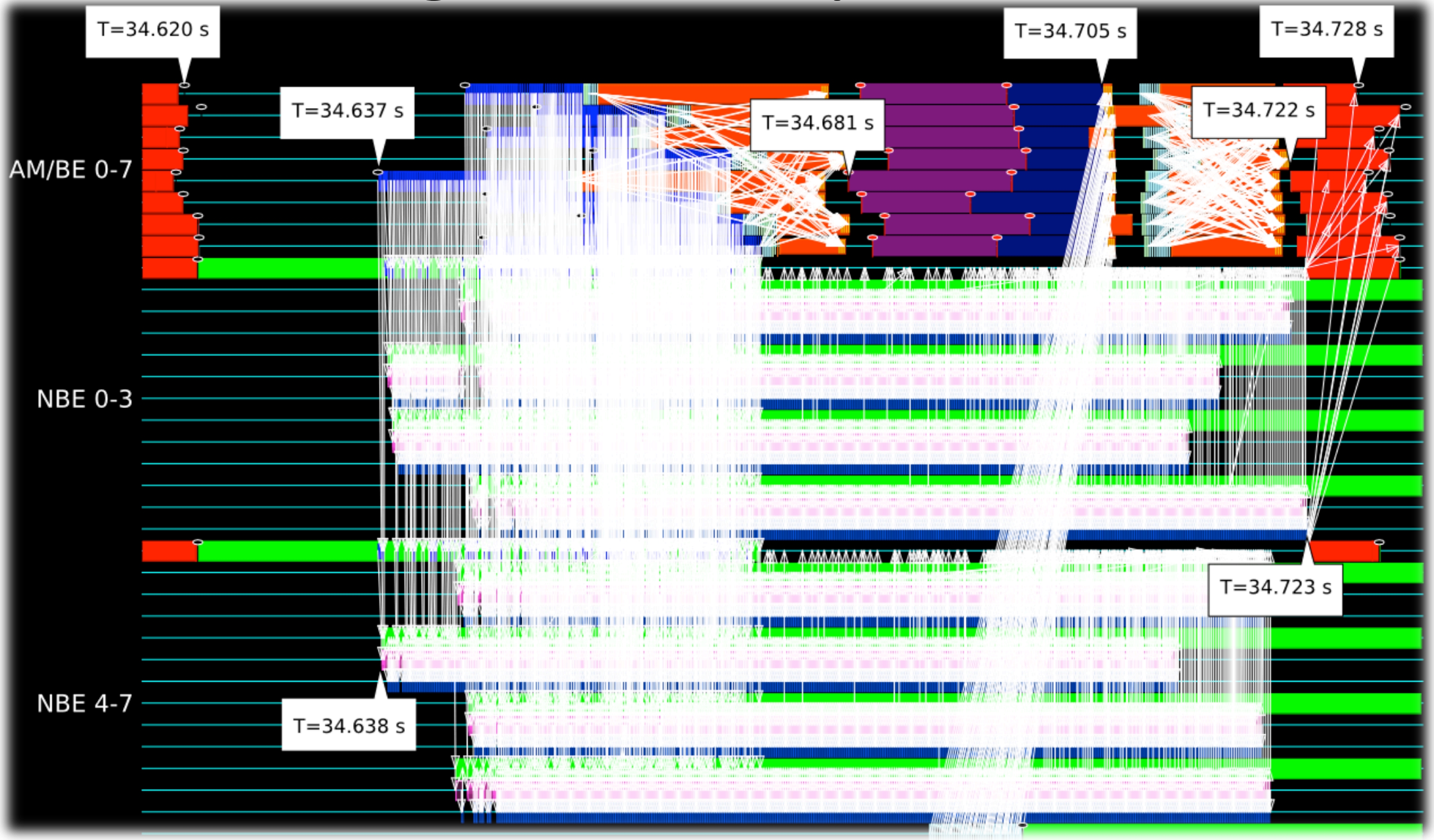
Problem :
Difficult to mathematically predict the expected speedup *a priori* due to the contentious nature of many-to-many communications.

Solution:
Measuring the non-deterministic behaviour using Jumpshot on the software version and back-annotate the deterministic behaviour.

- Make use of existing tools!



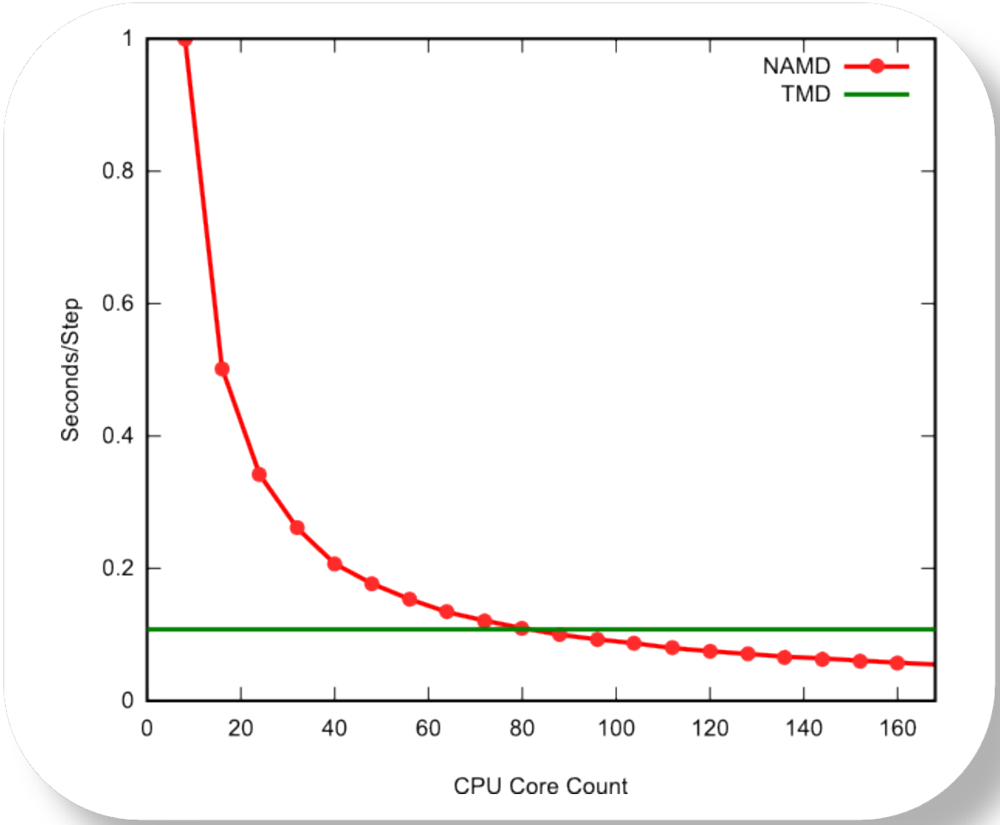
Single Timestep Profile



Timestep = 108 ms (327 506 atoms)

Performance

- Significant overlap between all force calculations.
- 108.02 ms is equivalent to between 80 and 88 Infiniband-connected cores at U of T's supercomputer, SciNet.
- 160-176 hyperthreaded cores
- Can we do better?
  - 140 with hardware bond engines – change engine from SW to HW, no architectural change
  - More with QPI systems



Final Performance Equivalent for MD

	FPGA/CPU	Supercomputer	Scaling Factor
Space	5U	17.5*2U	1/7
Cooling	N/A	Share of 735-ton chiller	∞?
Capital Cost	$15000*	$120000	1/8
Annual Electricity Cost	$241 (Assuming 500W)	$6758	1/30
Performance (Core Equivalent)	140 Cores	1*140 Cores	140x

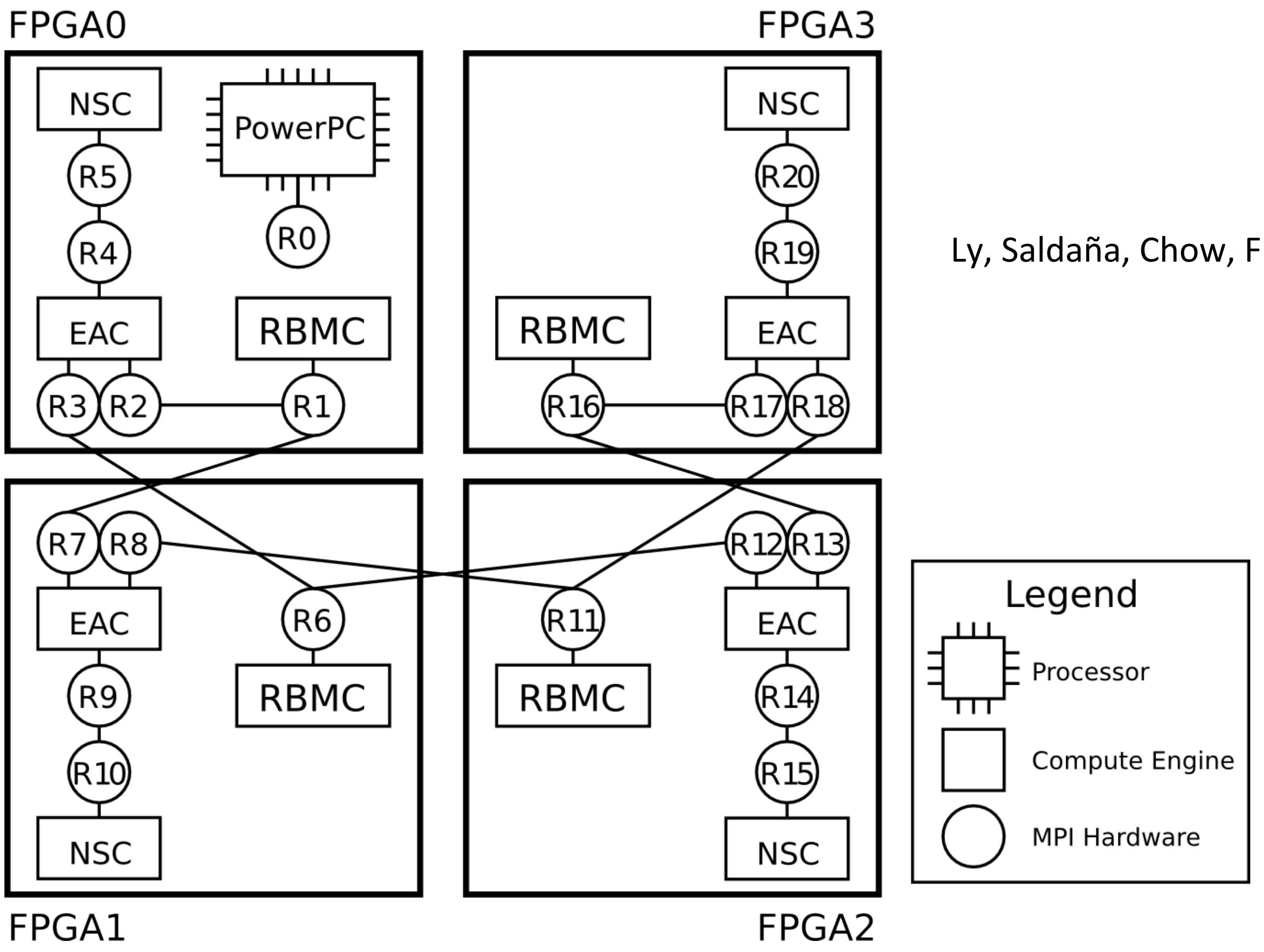
*Current system is a prototype. Cost is based on projections for next-generation system.

TMD Perspective

- Still comparing apples to oranges.
- Individually, hardware engines are able to sustain calculations hundreds of times faster than traditional CPUs.
- Communication costs degrade overall performance.
- FPGA platform is using older CPUs and older communication links than SciNet.
- Migrating the FPGA portion to a SciNet compatible platform will further increase the relative performance and provide a more accurate CPU/FPGA comparison.

BUILDING AN EMBEDDED APPLICATION

Restricted Boltzmann Machine



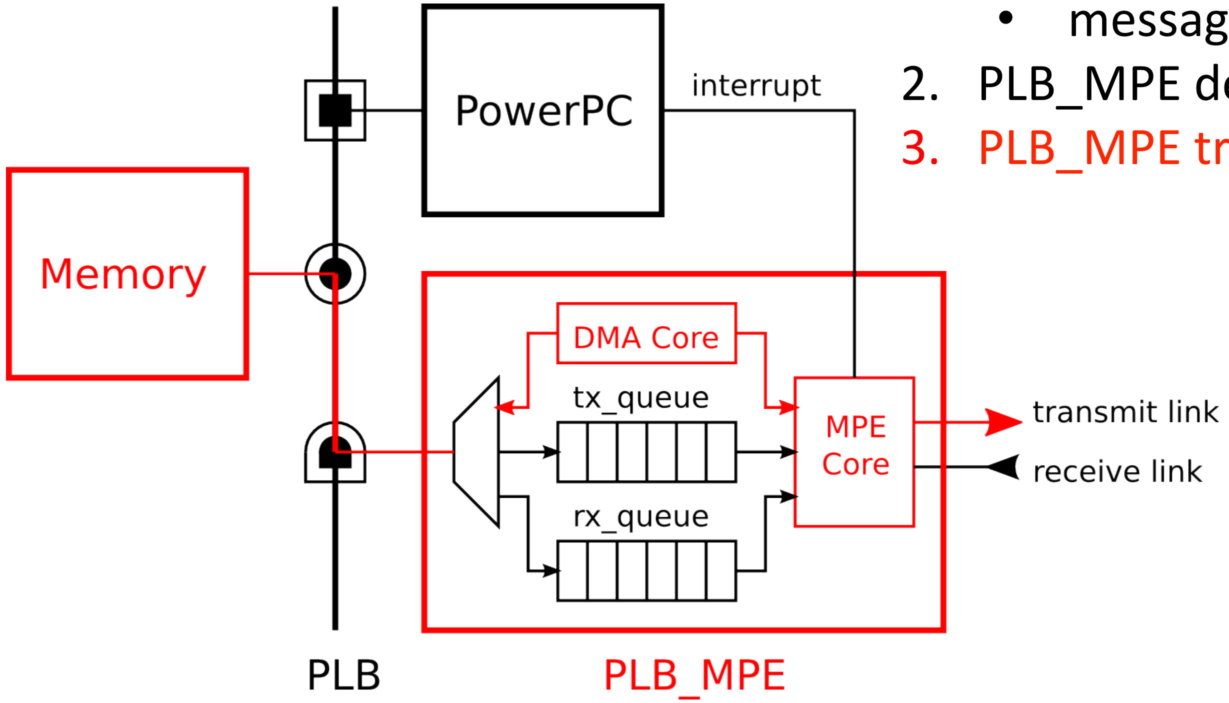
Ly, Saldaña, Chow, FPT 2009

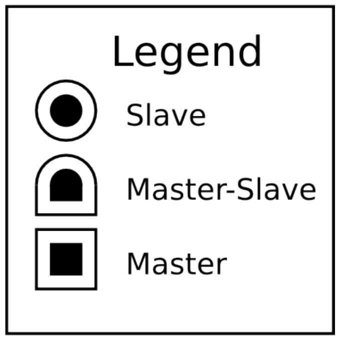
Class II: Processor-based Optimizations

Direct Memory Access MPI Engine

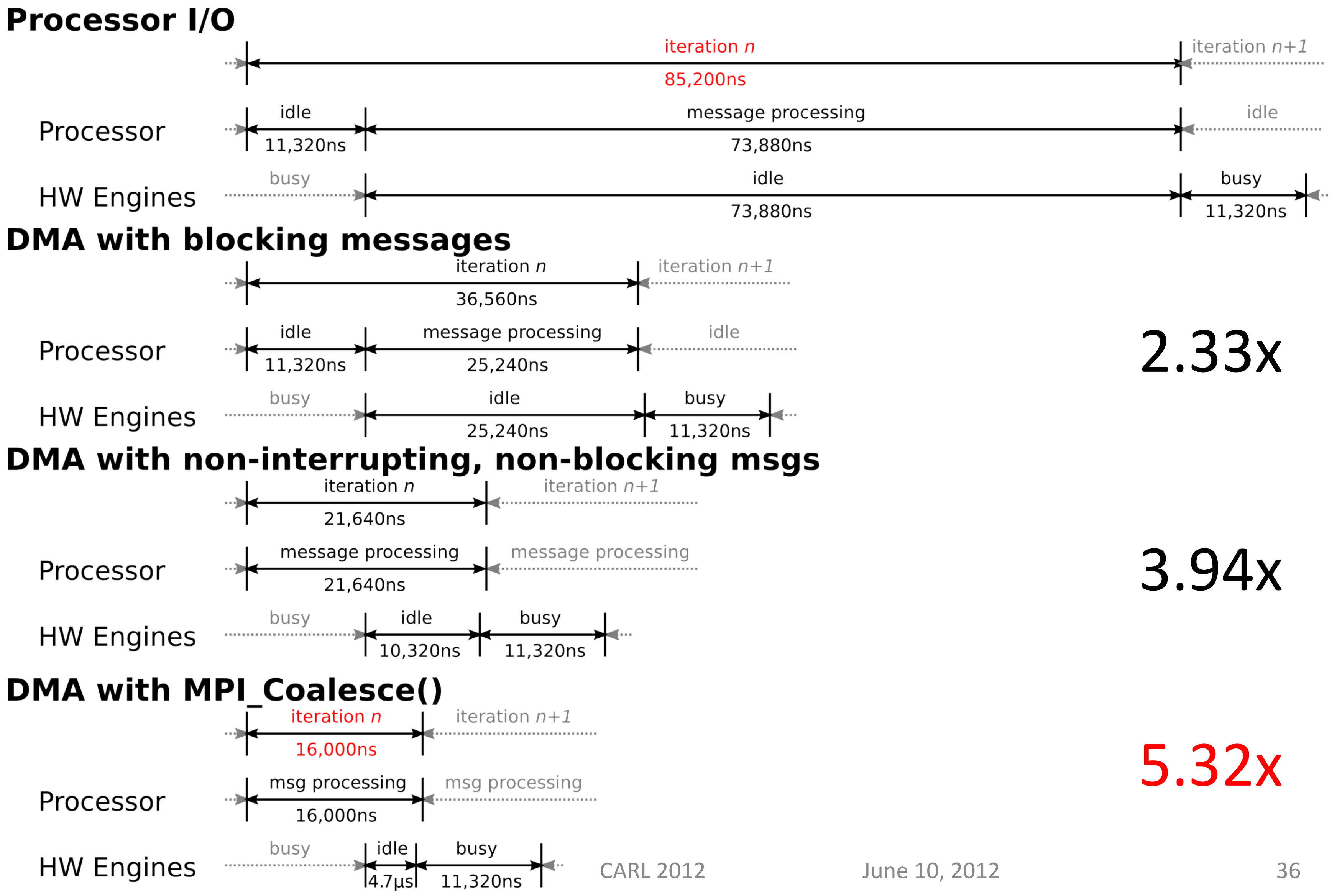
MPI_Send(...)

1. Processor writes 4 words
   - destination rank
   - address of data buffer
   - message size
   - message tag
2. PLB_MPE decodes message header
3. PLB_MPE transfers data from memory





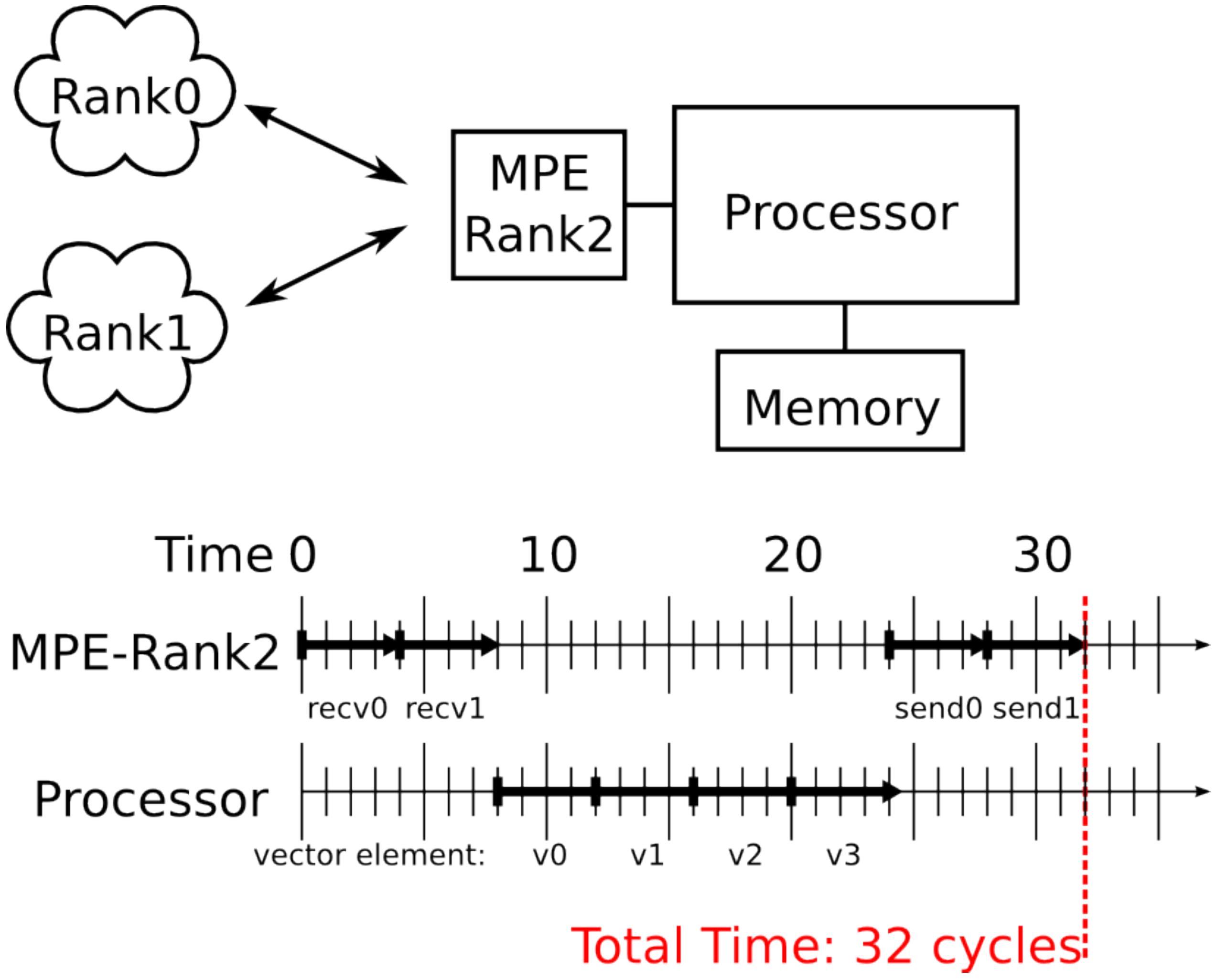
SW/HW Optimization



Class III: HW/HW Optimization

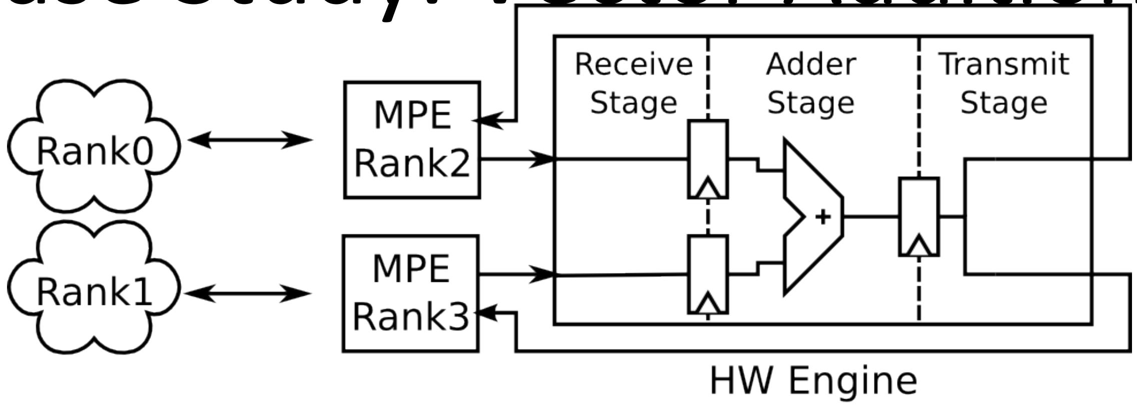
Messages vs Streams

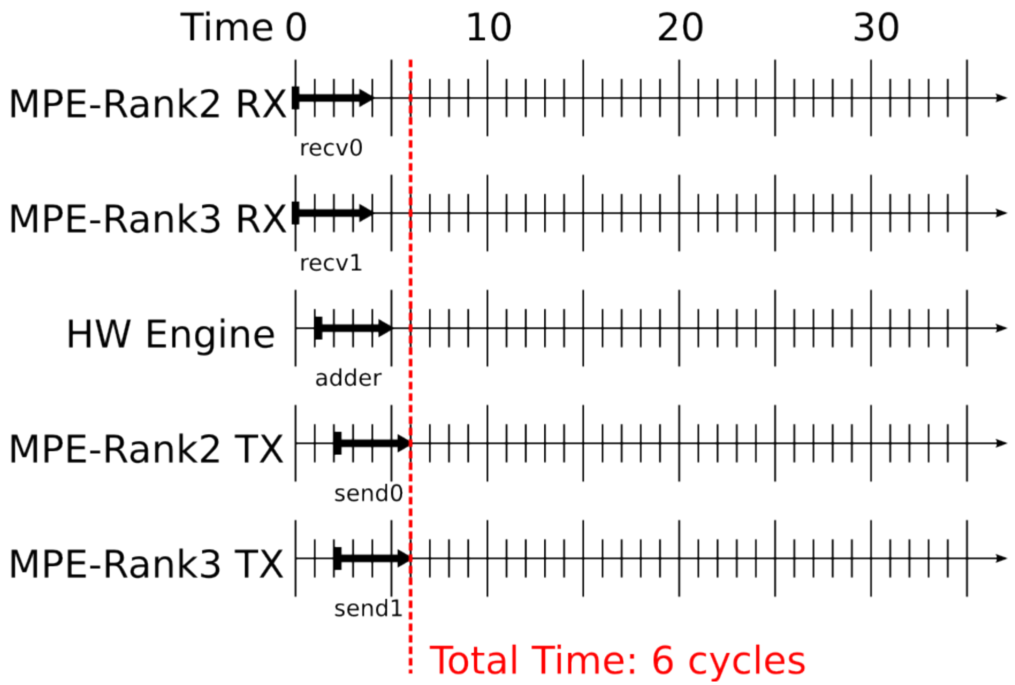
Case Study: Vector Addition



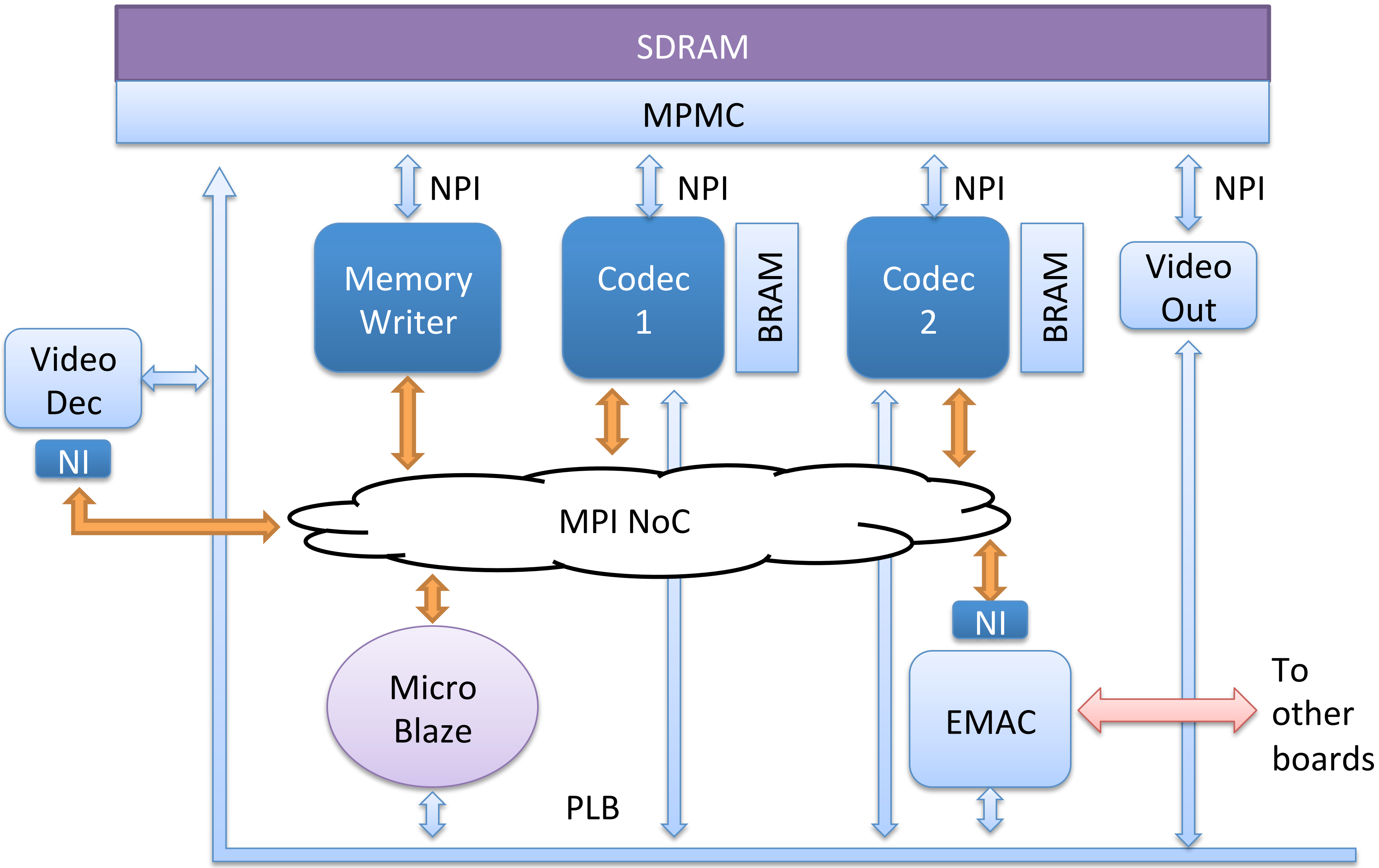
HW/HW Optimization

Case Study: Vector Addition

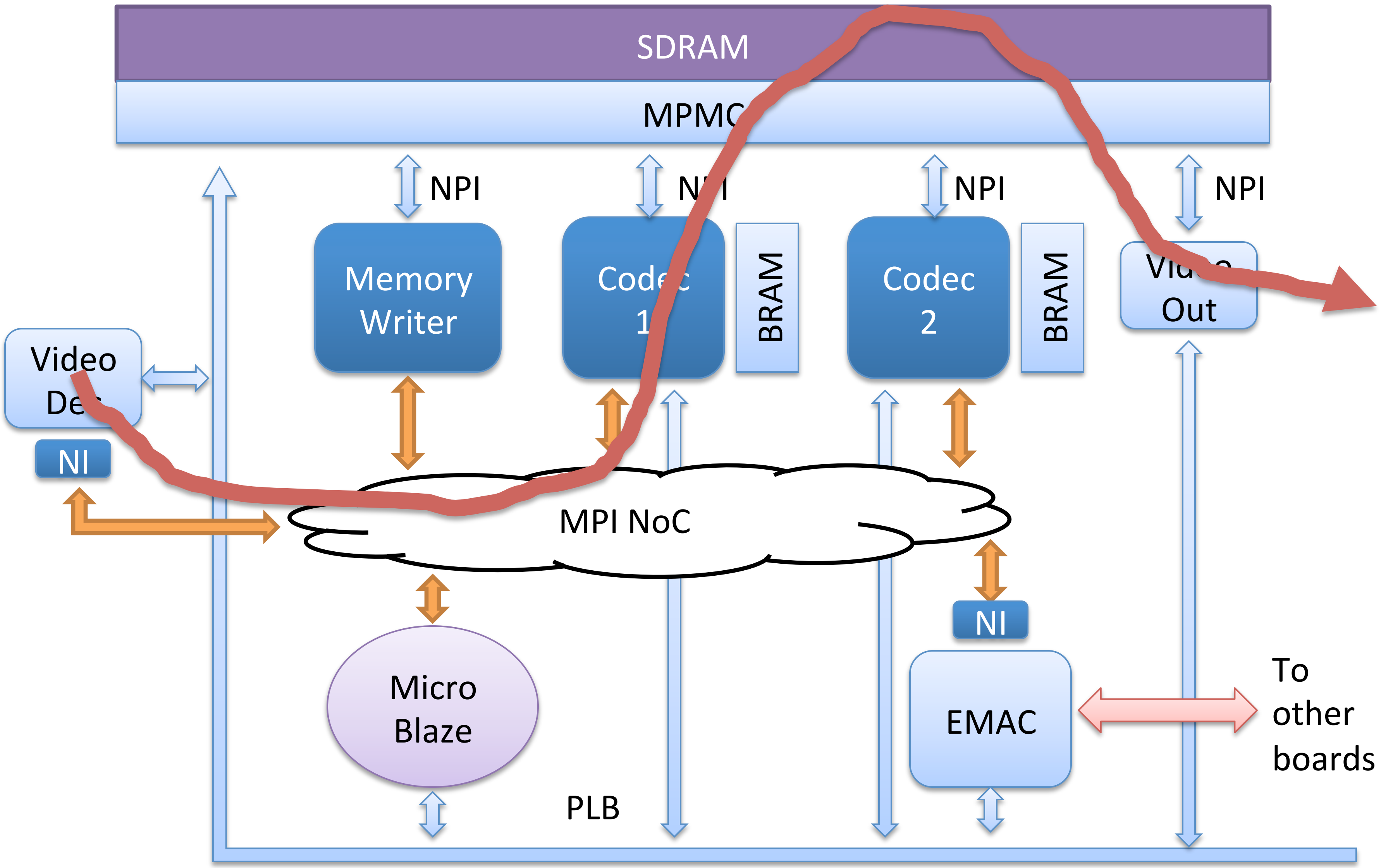




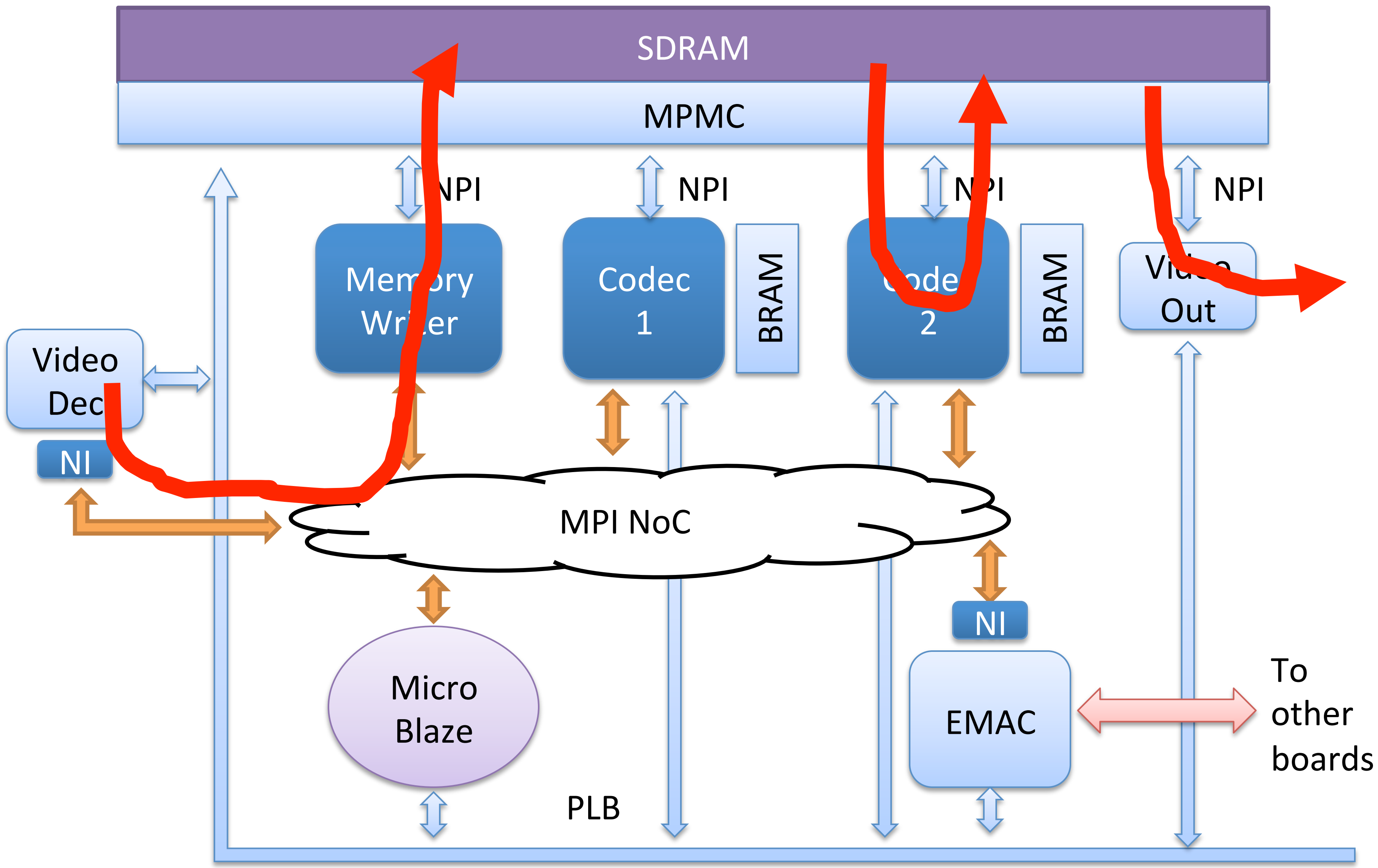
Scalable Video Processor (SoC)



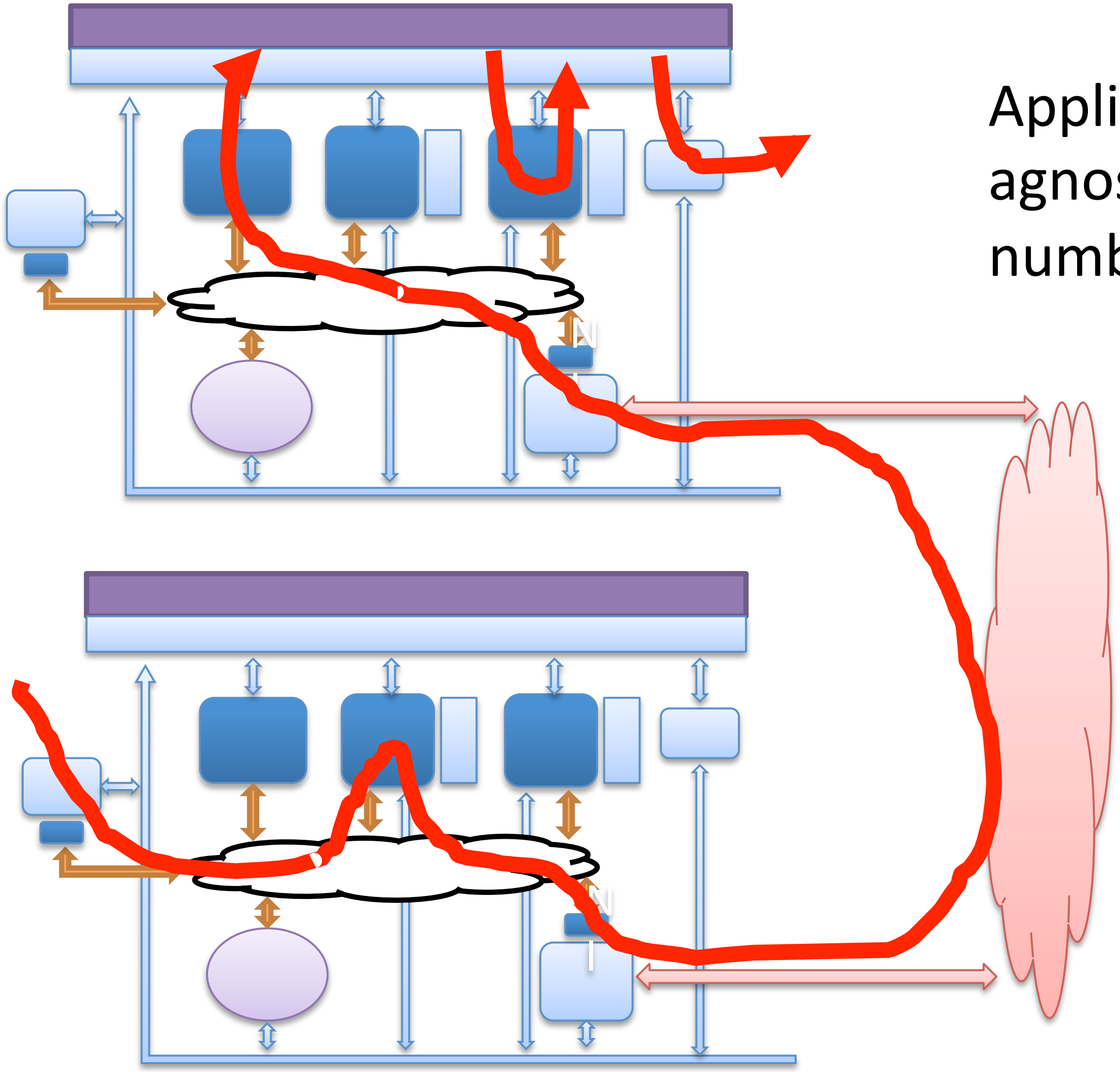
Streaming Codec



Frame Processing



Multi-Card System



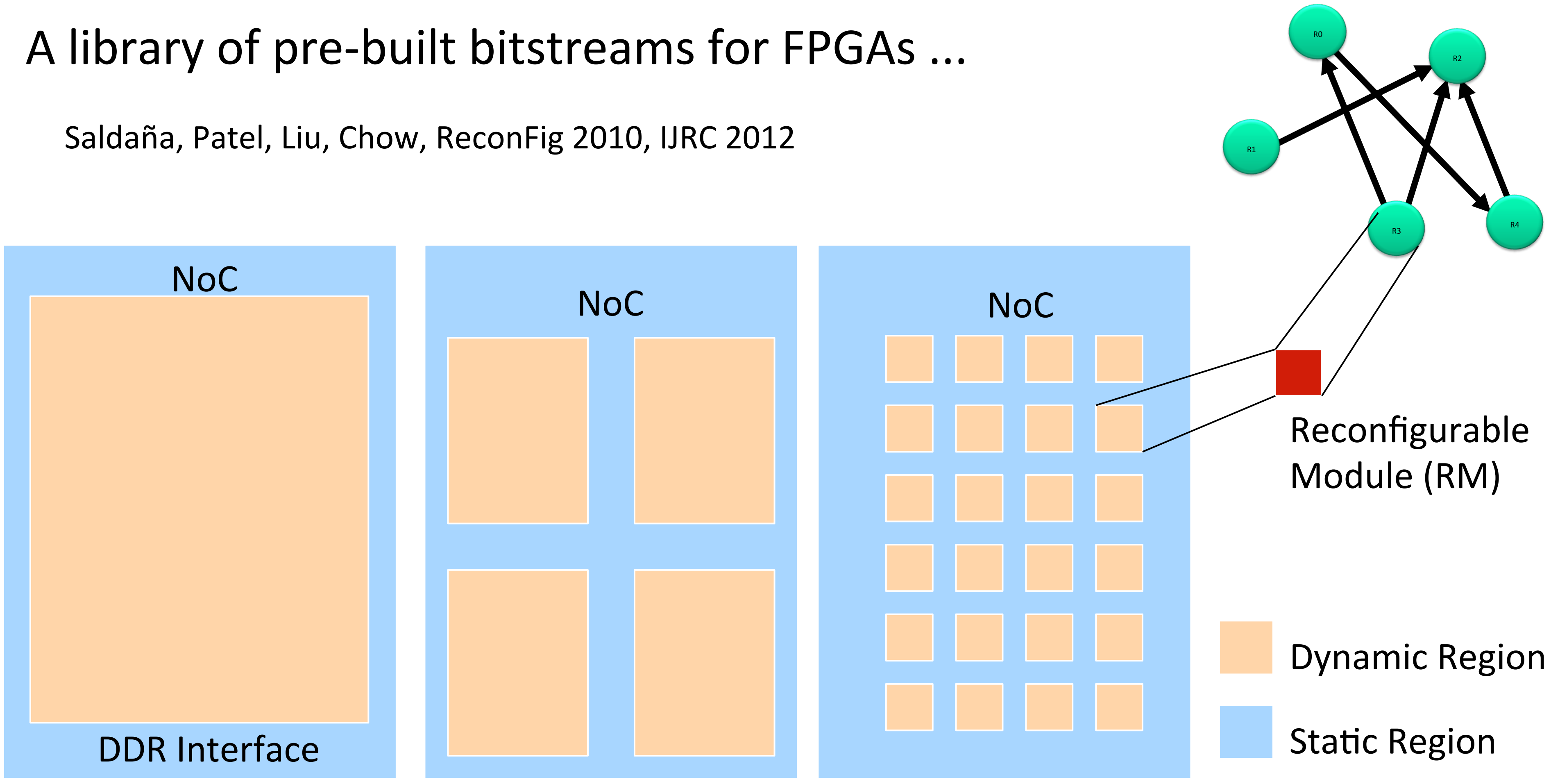
Application is agnostic to the number of cards

SUPPORTING PARTIAL RECONFIGURATION

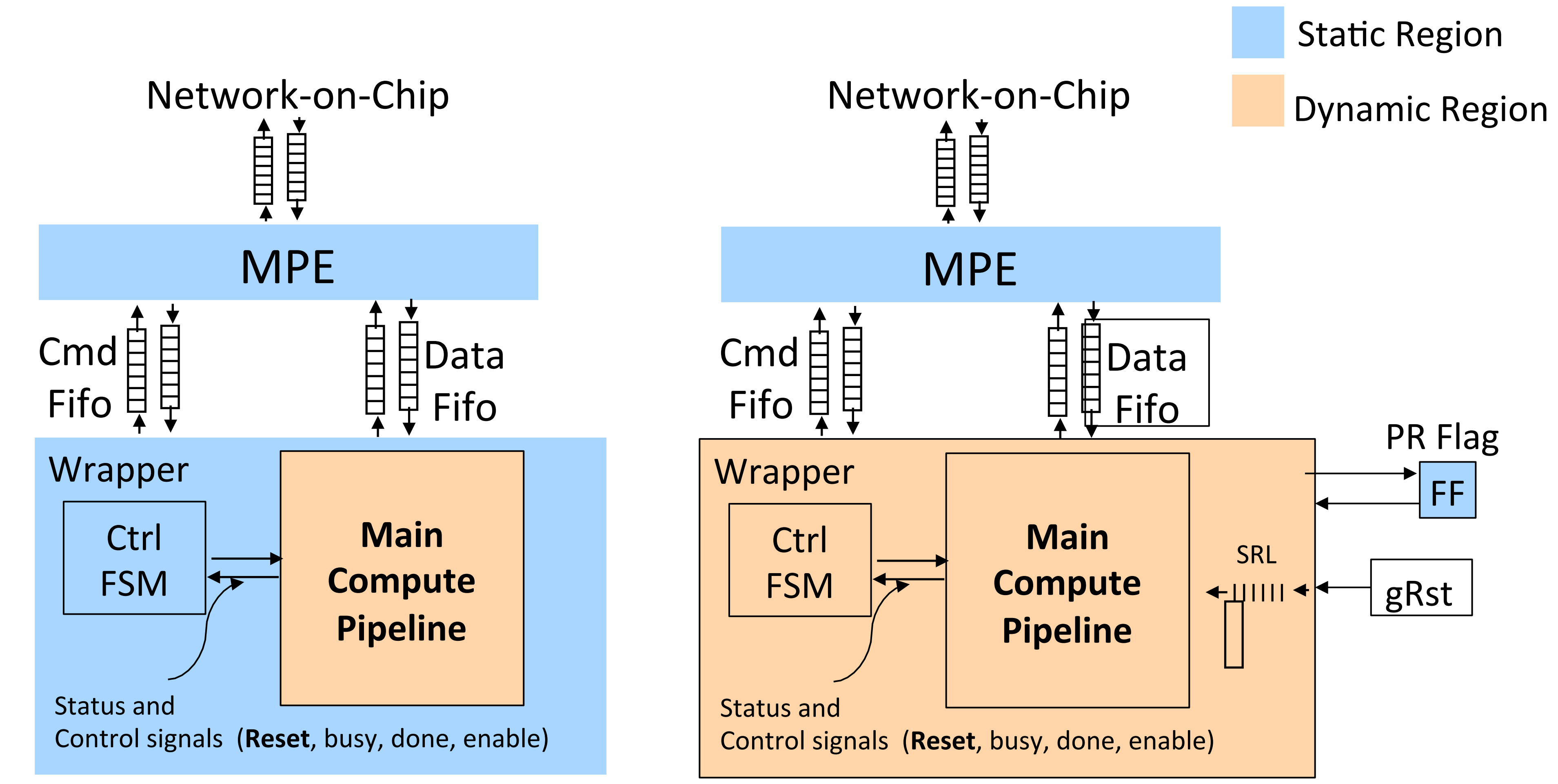
Template-based bitstreams

A library of pre-built bitstreams for FPGAs ...

Saldaña, Patel, Liu, Chow, ReconFig 2010, IJRC 2012



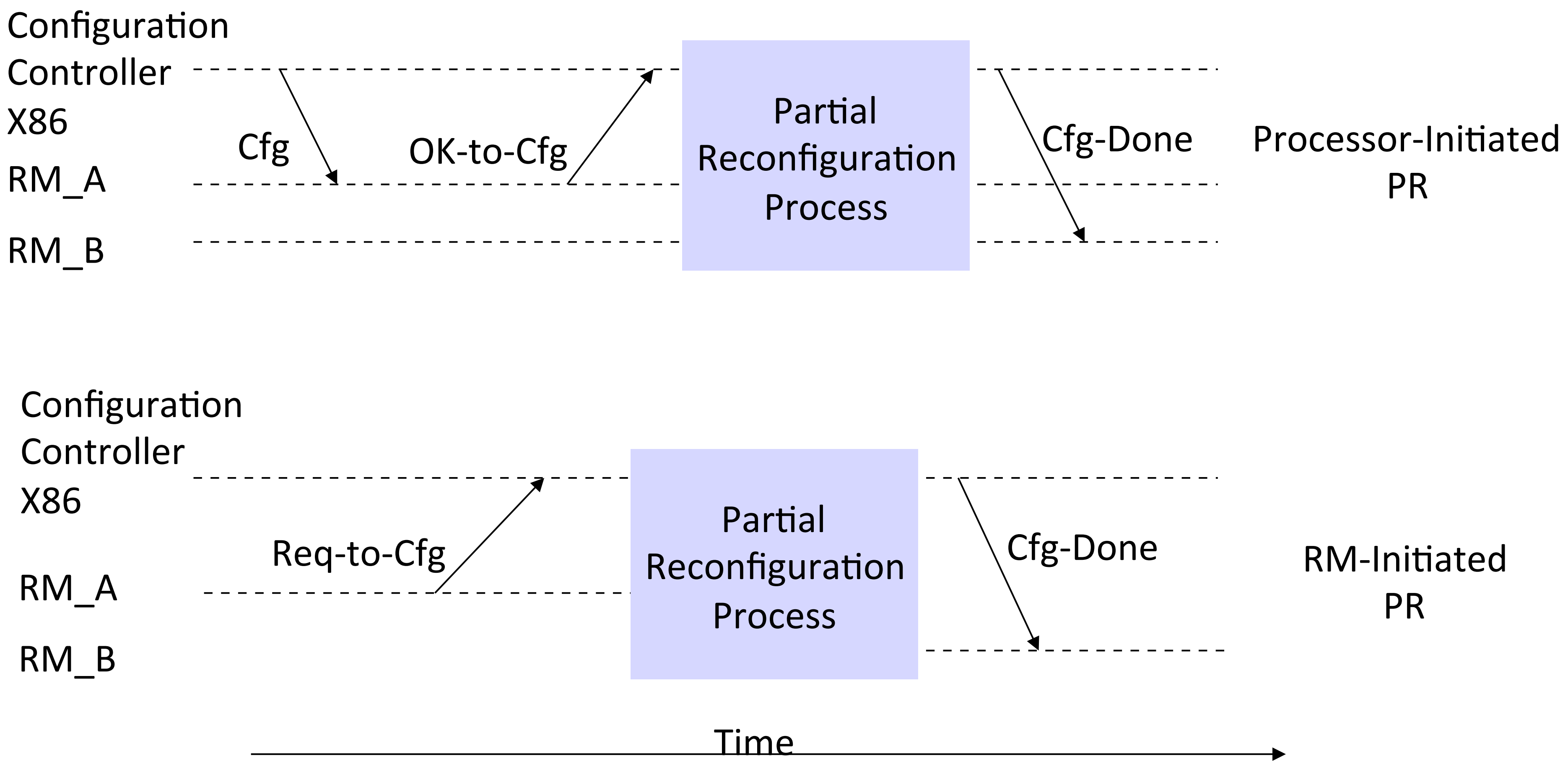
MPE and the Dynamic Region



Wrapper-contained
(Application-specific template bitstreams)

Self-contained
(Generic template bitstreams)

PR Synchronization, Data Store and Restore


Configuration Controller X86
RM_A
RM_B
Cfg
OK-to-Cfg
Partial Reconfiguration Process
Cfg-Done
Processor-Initiated PR
Configuration Controller X86
RM_A
RM_B
Req-to-Cfg
Partial Reconfiguration Process
Cfg-Done
RM-Initiated PR
Time

PR User code

```
MPI_Init();   // <--- Template bitstream configuration (where possible)
...
MPI_Send ( ..., dest, CFG_TAG,...);

MPI_Recv ( status_data_RM_A, … , dest,  OK_TO_CFG_TAG, ...);

ARCHES_MPI_Reconfig ( RM_B.bit, board_num, fpga_num );

MPI_Send ( status_data_RM_B, … , dest, CFG_DONE_TAG);
...
```

Processor-initiated PR

```
...
MPI_Recv ( status_data_RM_A, … , dest,  REQ_TO_CFG_TAG, ...);

ARCHES_MPI_Reconfig ( RM_B.bit, board_num, fpga_num );

MPI_Send ( status_data_RM_B, … , dest, CFG_DONE_TAG);
...
```

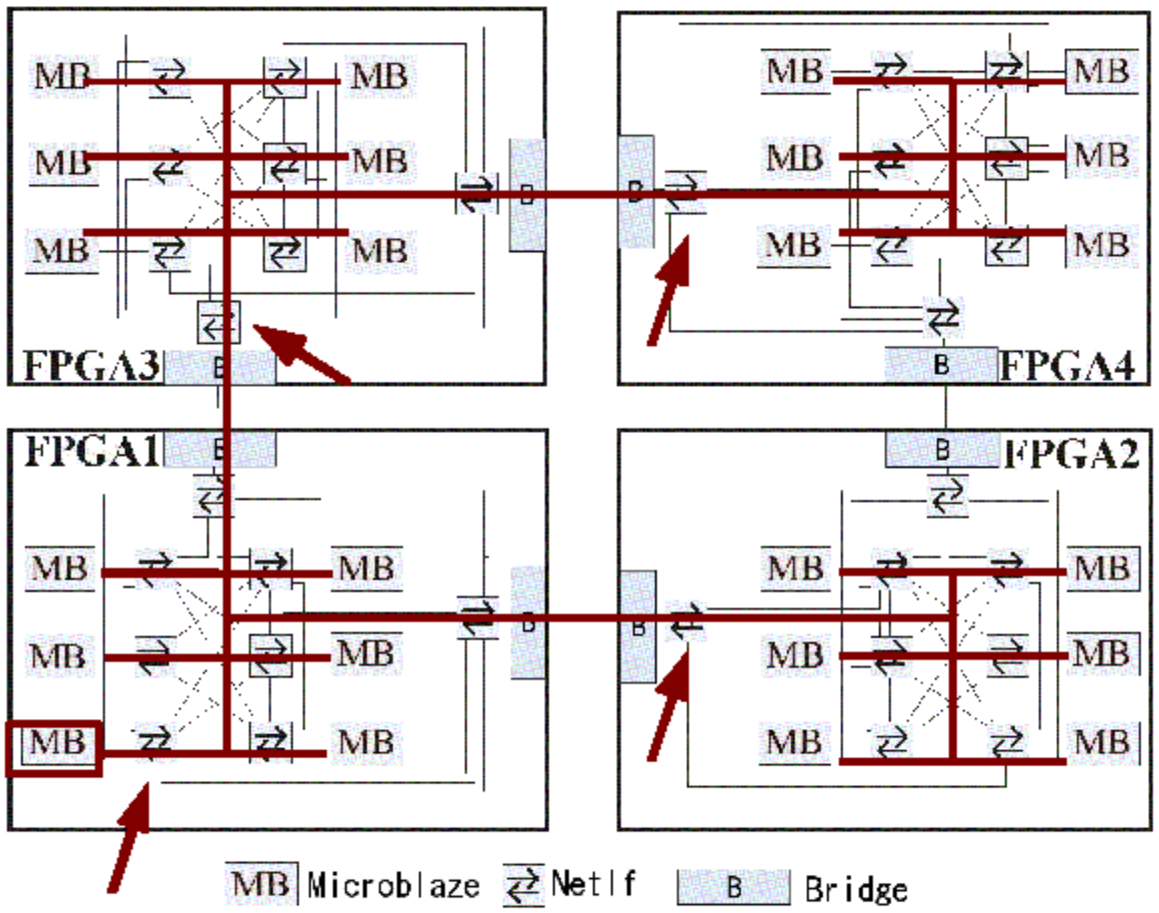
RM-initiated PR

HARDWARE SUPPORT FOR BROADCAST AND REDUCE

Experimental Platform

Peng, Saldaña, Chow, FPL2011

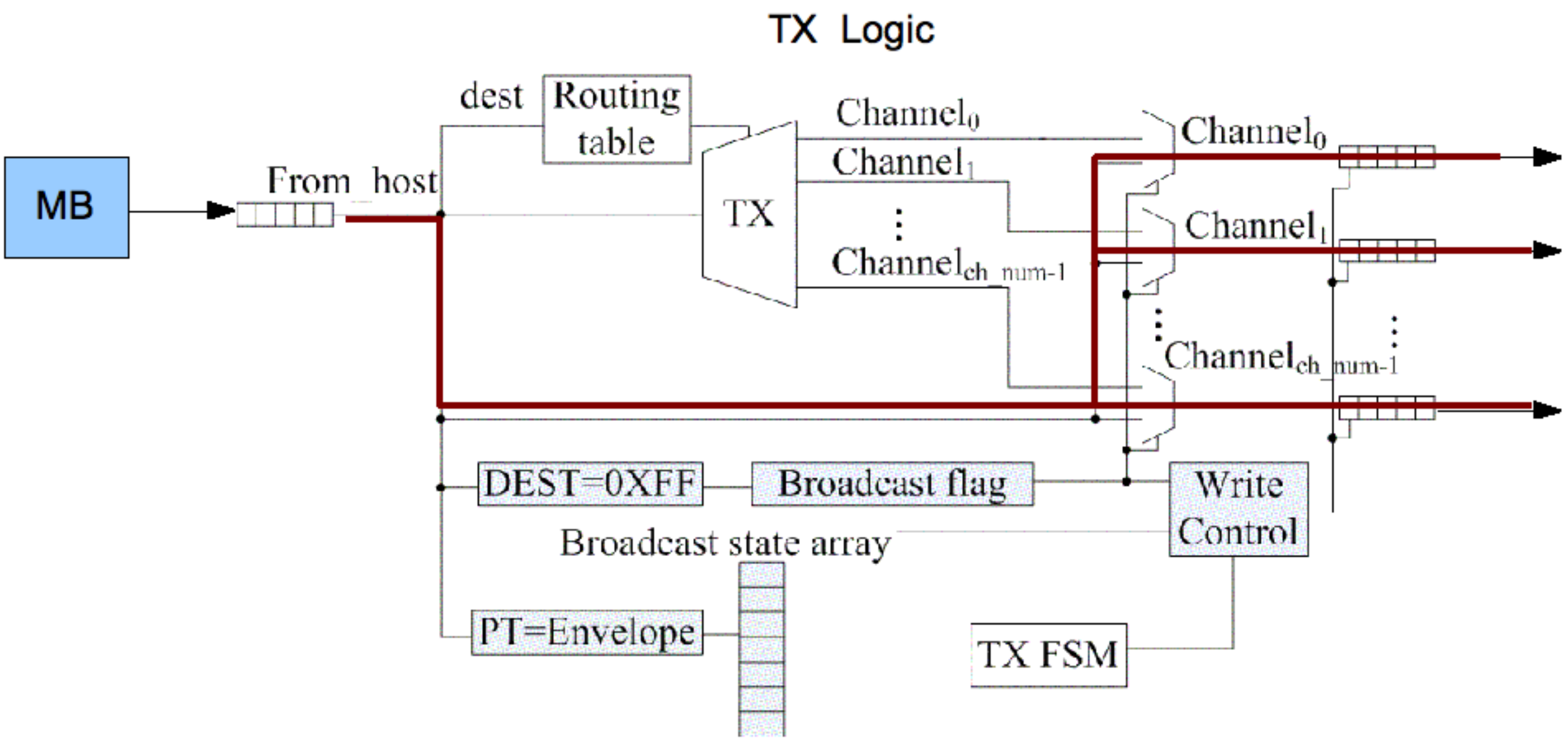
- 24-MicroBlaze System implemented on a BEE3 platform



Partial reductions

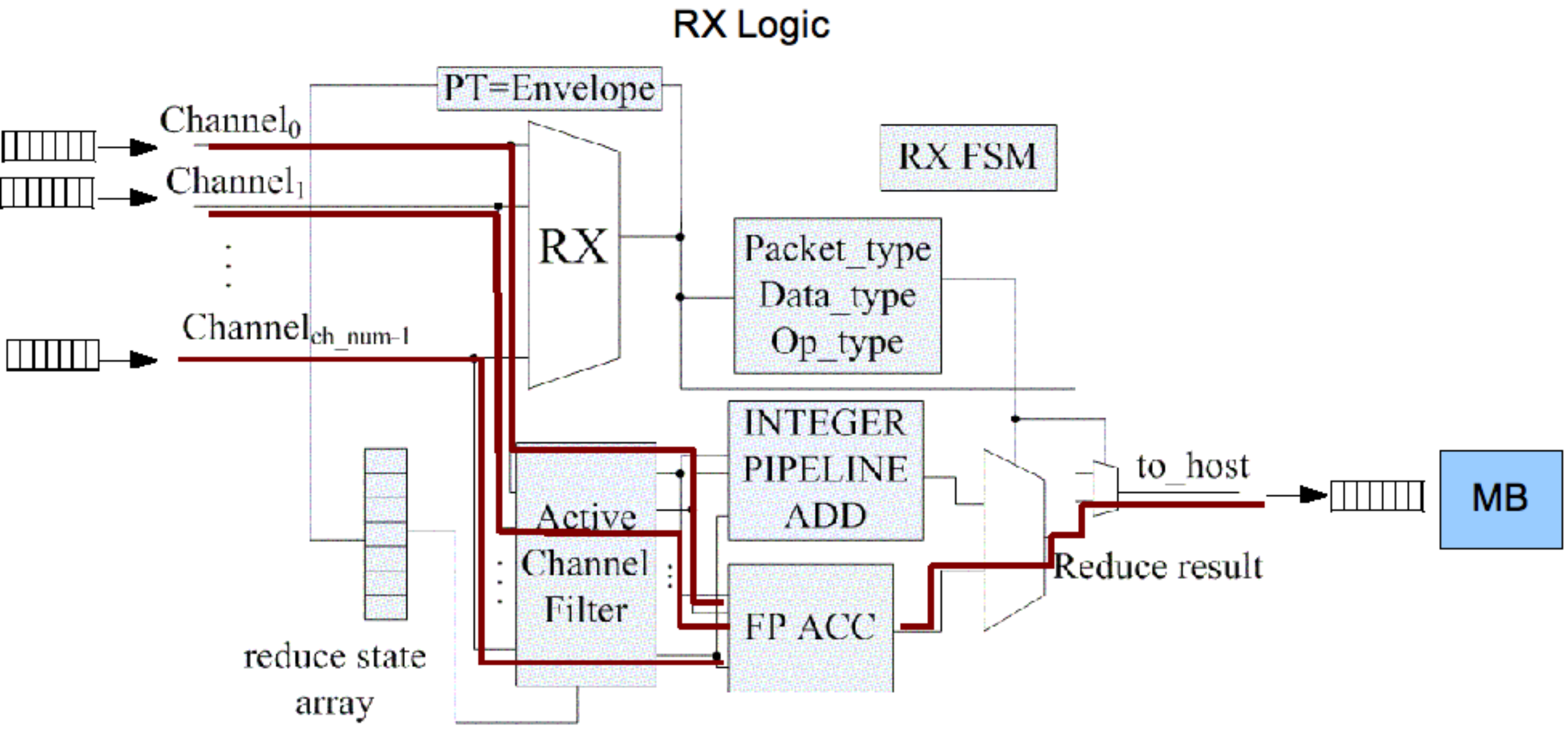
Changes to the NoC

- Broadcast block diagram



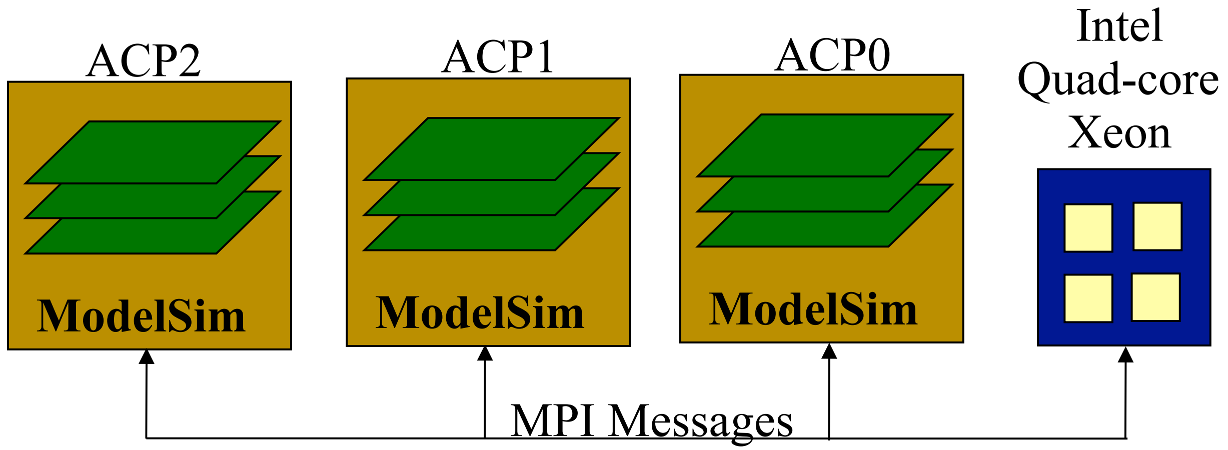
Changes to the NoC

- Reduce block diagram

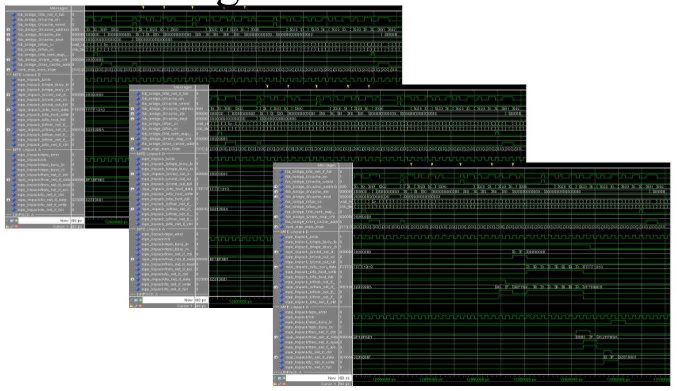


DEBUGGING AND PROFILING

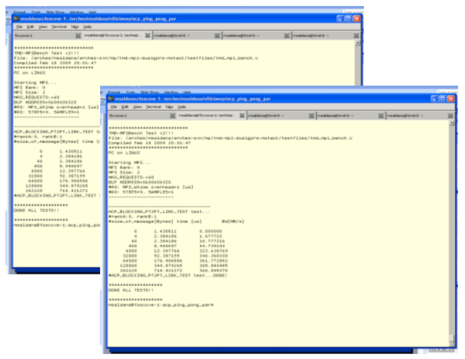
Debugging: Multi-FPGA System-Level Simulation



FPGA signals



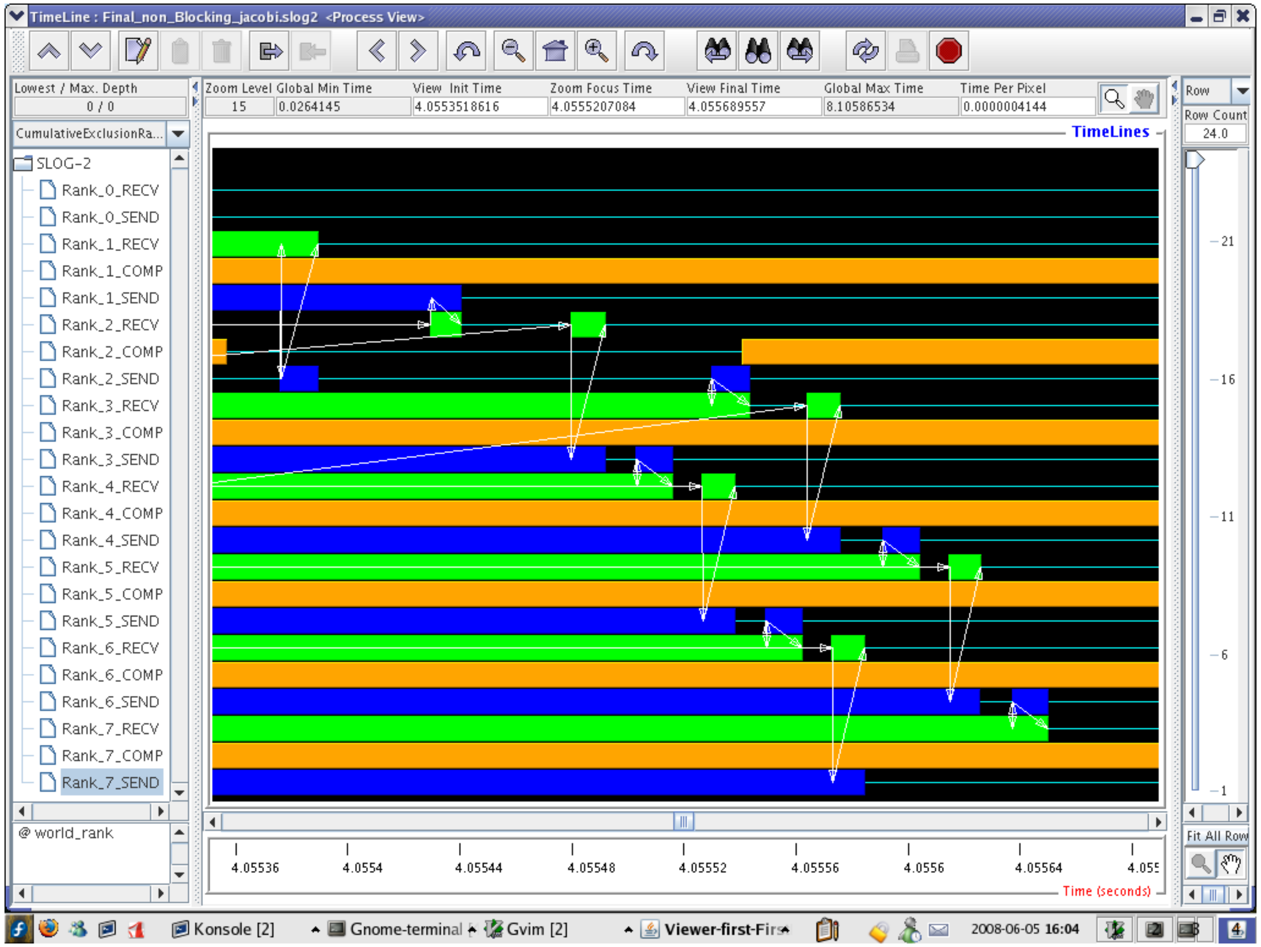
Stdout



- Test SW and HW ranks all at once
- No need to resynthesize
- Full visibility into the FPGA
- Good for modeling application-defined protocols at initial stages of development

Profiling: Jumpshot

Nunes, Saldaña, Chow, FPT 2008

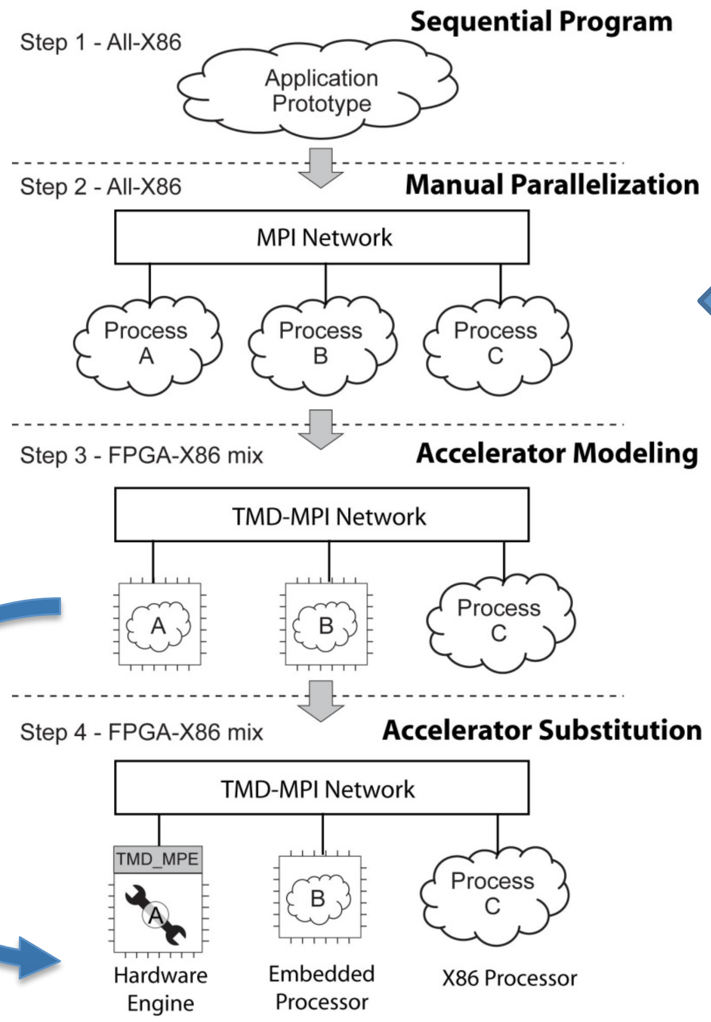

TimeLine : Final_non_Blocking_jacobi.slog2 <Process View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel	Row Count
0 / 0	15	0.0264145	4.0553518616	4.0555207084	4.055689557	8.10586534	0.0000004144	24.0

- Well-known tool
- Extracts MPI protocol states from the MPE
- Profile just like in Software
- Works only for embedded processors and hardware engines

ADDING HIGH-LEVEL SYNTHESIS

The Flow



Also a system simulation

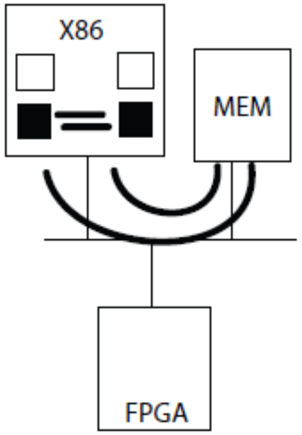
HLS
AutoESL

Benefits of Using MPI for HLS

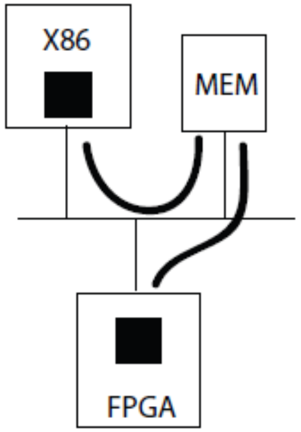
- High-level data and control flow defined at MPI message level
- Not synthesizing the entire system
  - HLS focus is on kernels of computation
- Interfaces are well-defined
  - Easier hand-off to HLS tool (or human)

SOME NUMBERS

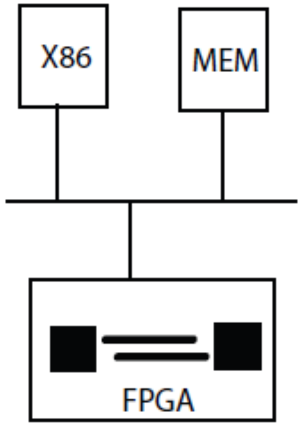
Configurations for Performance Testing



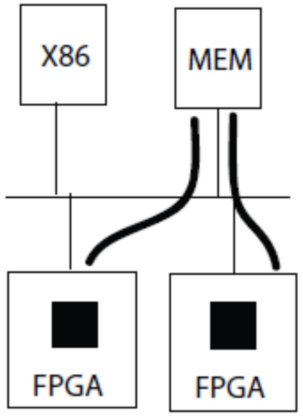
Xeon-Xeon



Xeon-HW



Intra-FPGA HW-HW



Inter-FPGA HW-HW

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

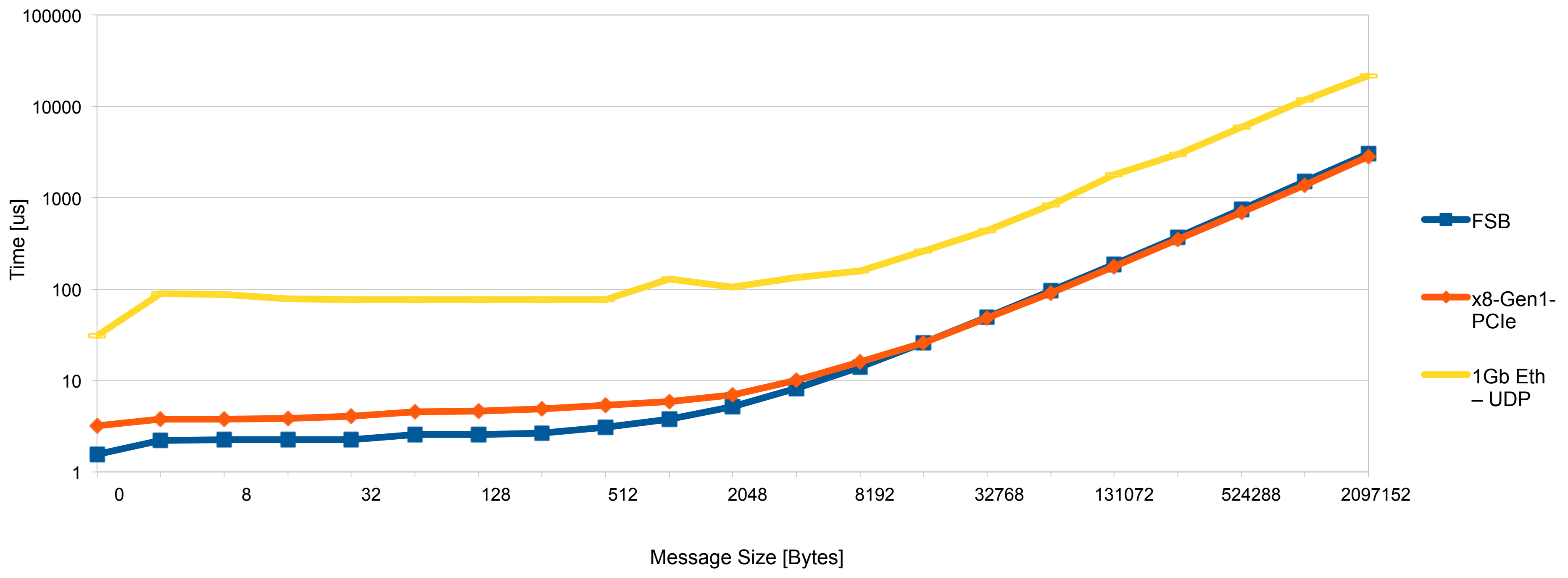
- MPI Ready Protocol achieves about 1/3 of the Rendezvous latency. For Xeon-HW it is 1μs (only 2X slower than Xilinx driver transfer latency)

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

Latency

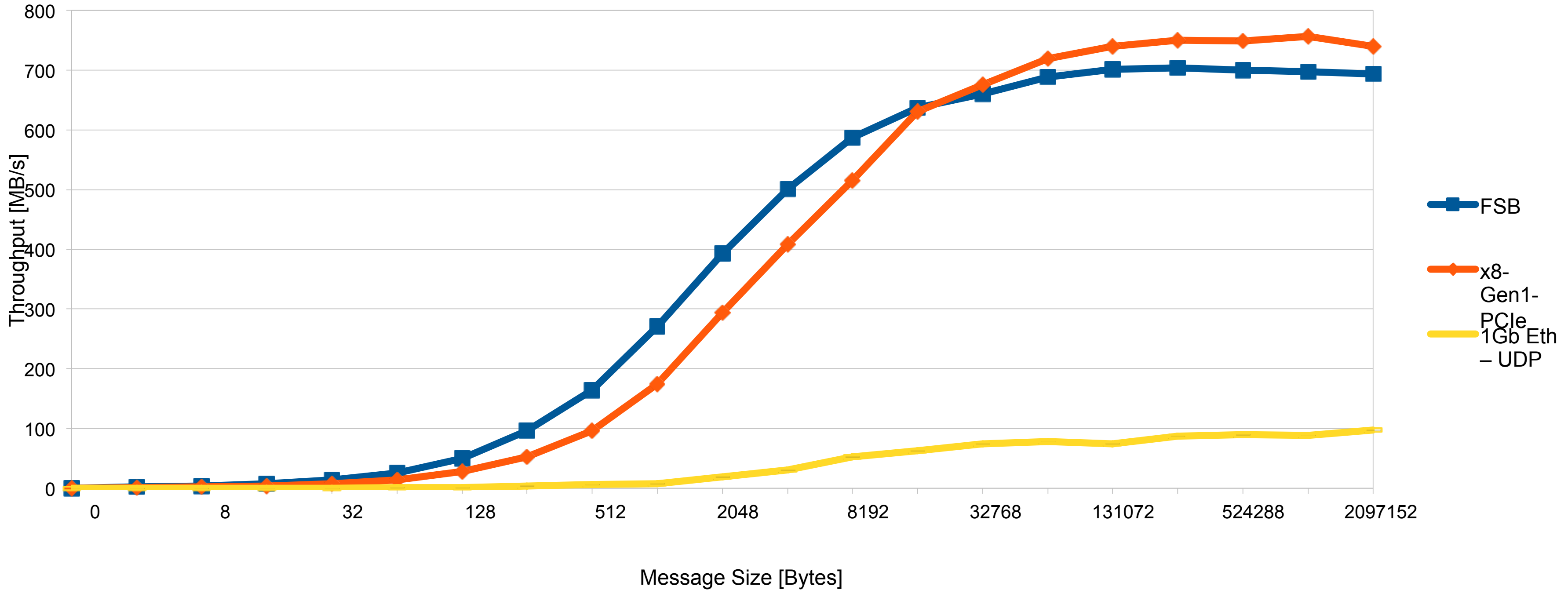
(delta = n^2)

Point-to-point with Rendezvous Protocol



Bandwidth

Point-to-point with Rendezvous Protocol



Conclusions I

- **Raising the level of abstraction important**
  - scalability, portability, flexibility, reusability, maintainability
  - productivity, accessible to application experts
- **Adapting an existing programming model brings an ecosystem that can be leveraged**
  - Debugging
  - Profiling
  - Knowledge and experience

Conclusions II

- **Adapting MPI works well**
  - Well-known programming model for parallel processing
  - Significant ecosystem for heterogeneous systems possible
  - Provides for incremental and iterative design
- **MPI can be easily extended/adapted for a heterogeneous environment**
  - Messages vs streaming
  - Coalescing
  - Partial reconfiguration

Conclusions III

- **Computational architecture may change with awareness of heterogeneous computing elements**
  - MD – heterogeneous versus homogeneous partitioning
  - Messages used to carry instructions to engines
- **Must do more top-down thinking about how to use/incorporate FPGAs into the computing world**
  - Mostly bottom-up (hardware) thinking so far
  - OpenCL looks to be a popular path today

The Real Workers

Taneem Ahmed
Xander Chin
Charles Lo
Chris Madill
Vince Mirian
Arun Patel
Manuel Saldaña
Kam Pui Tang
Ruediger Willenberg

Chris Comis
Danny Gupta
Alex Kaganov
Daniel Ly
Daniel Nunes
Emanuel Ramalho
Lesley Shannon
David Woods
Mike Yan

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