コンピューティングシステム・アーキテクチャに関する研究と展開

吉瀬 謙二
東京工業大学 大学院情報理工学研究科

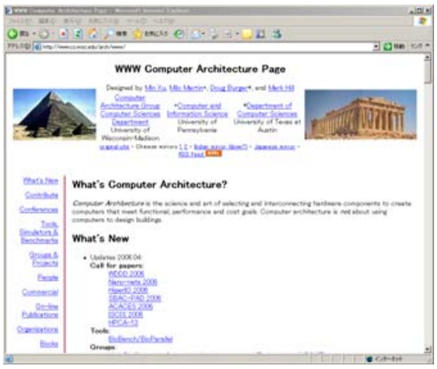
K. Kise TOKYO TECH 2015-01-19 13:00 – 14:30

Background

Computer Architecture

What's Computer Architecture?

Computer Architecture is the science and art of selecting and interconnecting hardware components to create **computers** that meet functional, performance and cost goals. Computer architecture is *not* about using computers to design buildings.



WWW Computer Architecture Page

Designed by Min Xu, Milo Martin*, Doug Burger*, and Mark Hill
Computer Architecture Group Computer Sciences Department University of Wisconsin-Madison
*Computer and Information Science University of Pennsylvania
*Department of Computer Sciences University of Texas at Austin
original site - Chinese mirrors 1 2 - Indian mirror (slow?) - Japanese mirror - RSS Feed

What's Computer Architecture?

Computer Architecture is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. Computer architecture is *not* about using computers to design buildings.

What's New

- Updates 2006.04:
  Call for papers:
  WDDD 2006
  Nano-nets 2006
  HiperIO 2006
  SBAC-PAD 2006
  ACACES 2006
  ISCIS 2006
  HPCA-13
  Tools:
  BioBench/BioParallel
  Groups:

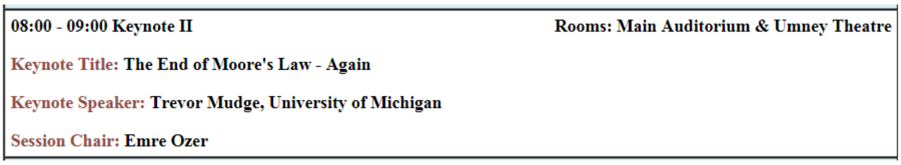
http://www.cs.wisc.edu/arch/www/

IPSJ SIG-ARC (The IPSJ Special Interest Group on Computer Architecture)
This April, its' name will change from **Computer Architecture** to **System Architecture**.

情報処理学会
計算機アーキテクチャ研究会
The IPSJ Special Interest Group on Computer Architecture

Background

- International Symposium on Microarchitecture (MICRO) 2014
- Keynote II

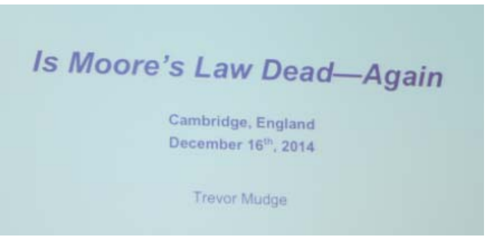


08:00 - 09:00 Keynote II — Rooms: Main Auditorium & Umney Theatre

Keynote Title: The End of Moore's Law - Again

Keynote Speaker: Trevor Mudge, University of Michigan

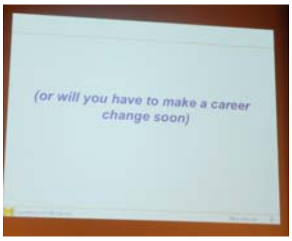
Session Chair: Emre Ozer



Is Moore's Law Dead—Again

Cambridge, England
December 16th, 2014

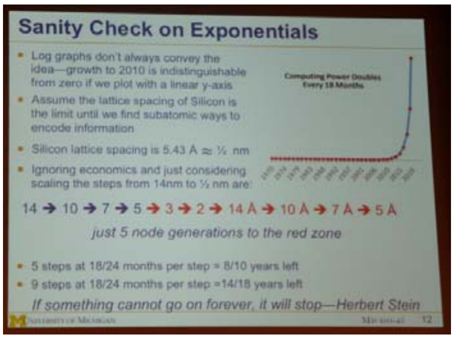
Trevor Mudge



(or will you have to make a career change soon)

Background

- *Is Moore's Low Dead - Again*



Sanity Check on Exponentials

- Log graphs don't always convey the idea—growth to 2010 is indistinguishable from zero if we plot with a linear y-axis
- Assume the lattice spacing of Silicon is the limit until we find subatomic ways to encode information
- Silicon lattice spacing is 5.43 A ≈ ½ nm
- Ignoring economics and just considering scaling the steps from 14nm to ½ nm are:

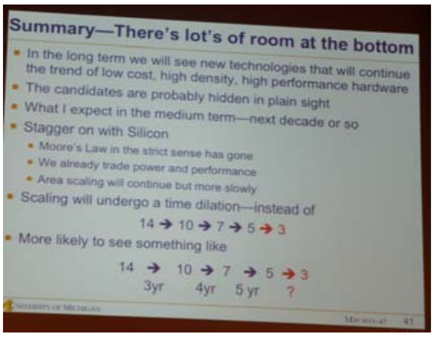
14 → 10 → 7 → 5 → 3 → 2 → 14 Å → 10 Å → 7 Å → 5 Å

just 5 node generations to the red zone

- 5 steps at 18/24 months per step = 8/10 years left
- 9 steps at 18/24 months per step =14/18 years left

If something cannot go on forever, it will stop—Herbert Stein

Computing Power Doubles Every 18 Months
UNIVERSITY OF MICHIGAN
12



Summary—There's lot's of room at the bottom

- In the long term we will see new technologies that will continue the trend of low cost, high density, high performance hardware
- The candidates are probably hidden in plain sight
- What I expect in the medium term—next decade or so
- Stagger on with Silicon
  - Moore's Law in the strict sense has gone
  - We already trade power and performance
  - Area scaling will continue but more slowly
- Scaling will undergo a time dilation—instead of

14 → 10 → 7 → 5 → 3

- More likely to see something like

14 → 10 → 7 → 5 → 3
3yr 4yr 5 yr ?

Background

- International Symposium on Microarchitecture (MICRO) 2014
- Hot-Debate

18:00 - 19:15 Hot-Debate **Place: Cambridge Union Debating Chamber**

Moderator: Trevor Mudge

Debate Topic: It is the End of the Road for the von Neumann Architecture

No - 149
Yes - 116

Those who spoke in favor of the motion were Simon Moore - Professor of Computer Engineering, The University of Cambridge and Fellow of Trinity Hall; Steve Furber - Professor of Computer Engineering, CBE and Fellow of the Royal Society; and Kunle Olukotun - Cadence Design Systems Professor of Electrical Engineering & Computer Science, Stanford University.

Those who spoke against the motion were Yale Patt - Professor of Electrical and Computer Engineering, Ernest Cockrell, Jr. Centennial Chair in Engineering, and University Distinguished Teaching Professor, The University of Texas at Austin; Simon Knowles - Chief Technology Officer, XMOS, Ltd.; and Guri Sohi - John P. Morgridge Professor and E. David Cronon Professor of Computer Sciences, The University of Wisconsin-Madison.

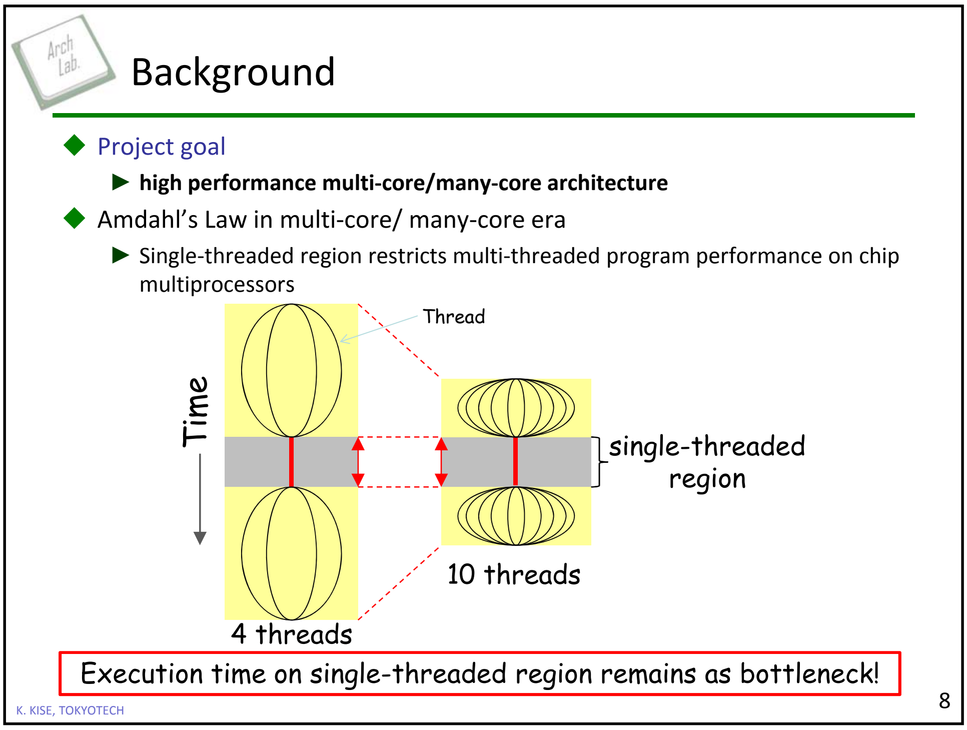
Agenda

- **Recent projects**
  - **CoreSymphony: Efficient and Realistic Cooperative Core Architecture**
  - **ScalableCore System: Scalable Many-core Emulator**
  - Challenges for Dependable Many-Core Processors
- **Ongoing projects**
  - **ArchHDL: A novel Hardware RTL Design Environment in C++**
  - **High-speed FPGA Accelerator for Integer Sorting**
  - **Fast and Accurate Emulation of Large-scale Network on Chip**
  - **MieruSys Project : Developing an Innovative Computer System**

CoreSymphony: Efficient and Realistic Cooperative Core Architecture

K. Kise TOKYO TECH 2015-01-19 13:00 – 14:30

K. KISE, TOKYOTECH

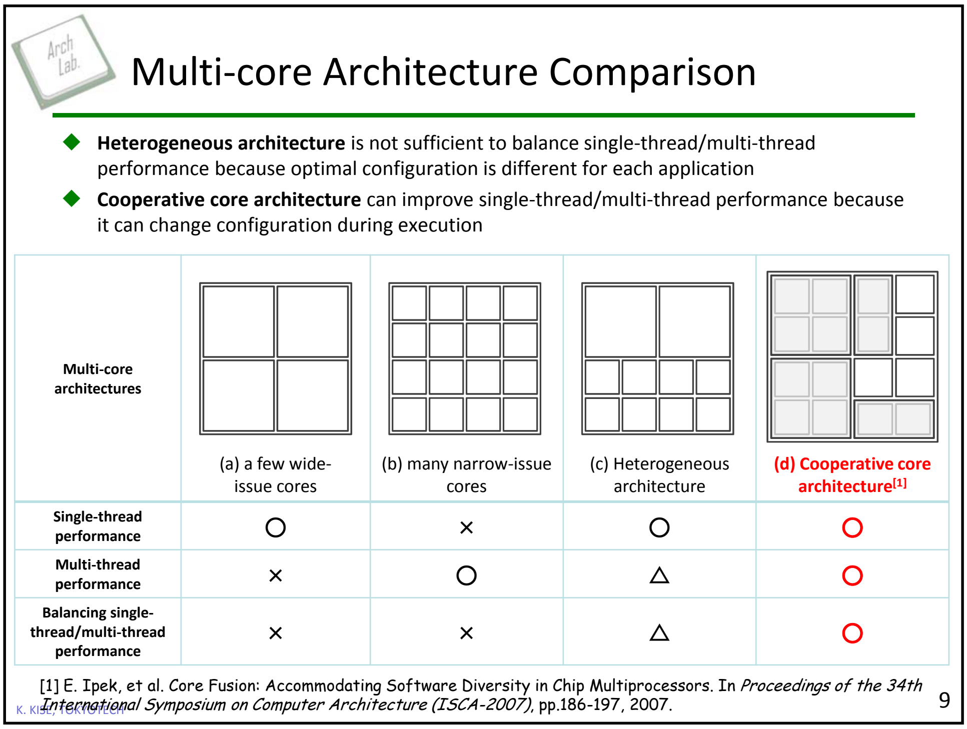


Background

- Project goal
  - **high performance multi-core/many-core architecture**
- Amdahl’s Law in multi-core/ many-core era
  - Single-threaded region restricts multi-threaded program performance on chip multiprocessors

Execution time on single-threaded region remains as bottleneck!

K. KISE, TOKYOTECH 8

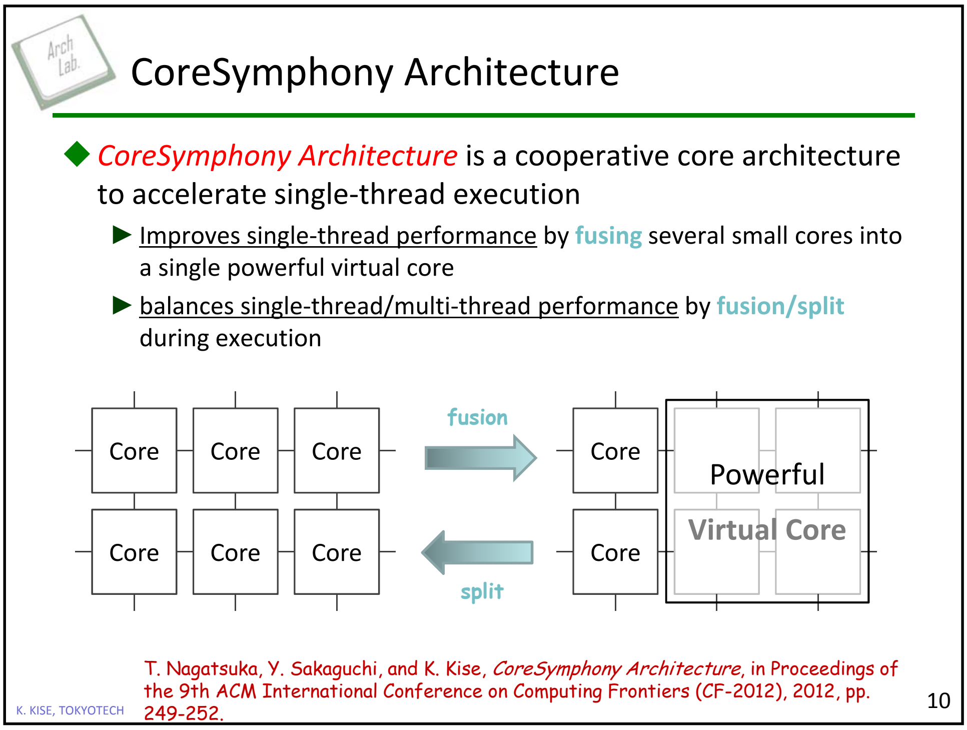


Multi-core Architecture Comparison

- **Heterogeneous architecture** is not sufficient to balance single-thread/multi-thread performance because optimal configuration is different for each application
- **Cooperative core architecture** can improve single-thread/multi-thread performance because it can change configuration during execution

Multi-core architectures	(a) a few wide-issue cores	(b) many narrow-issue cores	(c) Heterogeneous architecture	(d) Cooperative core architecture[1]
Single-thread performance	○	×	○	○
Multi-thread performance	×	○	△	○
Balancing single-thread/multi-thread performance	×	×	△	○

[1] E. Ipek, et al. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA-2007)*, pp.186-197, 2007.



CoreSymphony Architecture

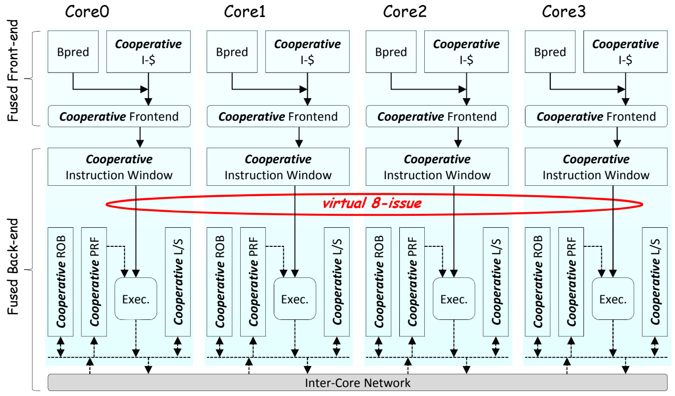
- *CoreSymphony Architecture* is a cooperative core architecture to accelerate single-thread execution
  - Improves single-thread performance by **fusing** several small cores into a single powerful virtual core
  - balances single-thread/multi-thread performance by **fusion/split** during execution

Core
Core
Core
Core
Core
Core
fusion
split
Core
Core
Powerful
Virtual Core

T. Nagatsuka, Y. Sakaguchi, and K. Kise, *CoreSymphony Architecture*, in Proceedings of the 9th ACM International Conference on Computing Frontiers (CF-2012), 2012, pp. 249-252.

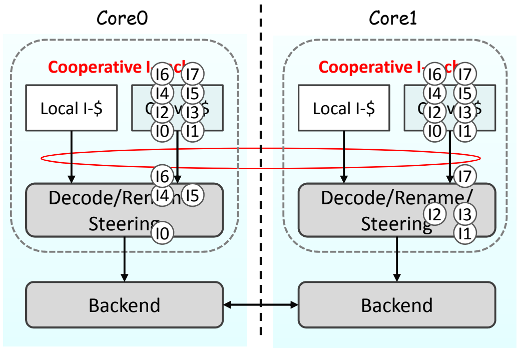
Differences between CoreSymphony and other cooperative core architectures

- Eliminates inter-core communications among front-ends to keep **core modularity**
- Does not require binary modification to keep **binary compatibility**
- Adopts 2-way out-of-order core as a baseline processor
- Supports up to **4-core fusion**, can construct up to 8-issue virtual core



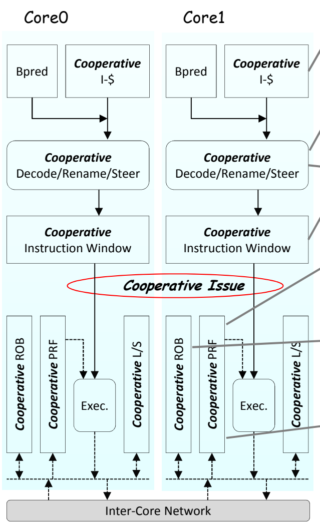
Conceptual diagram of CoreSymphony Architecture

How to fetch and execute instructions on a 2-core fusion virtual core



1. Fetch an instruction trace, called *Fetch Block* (FB), from **Conventional I-$**
2. Steer these instructions to fused cores (select the core which execute each inst.)
3. Execute steered instructions on each core (if necessary, results are broadcasted)
4. Store steered instruction to each **Local I-$**
5. If **Local I-$** hits, only steered instructions are fetched and executed

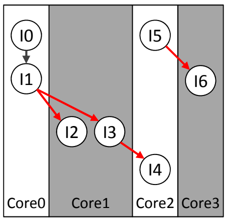
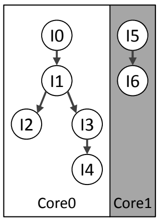
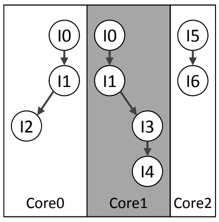
Problems and Proposal for CoreSymphony



- Instruction Cache
  - Local Instruction Cache
- Instruction Steering
  - Leaf-node Steering
- Register Renaming
  - 2-way Renaming
- Physical Register File (PRF)
  - PRF Distribution
- Reorder Buffer (ROB)
  - ROB Distribution
- In-order State Management
  - In-order State Distribution

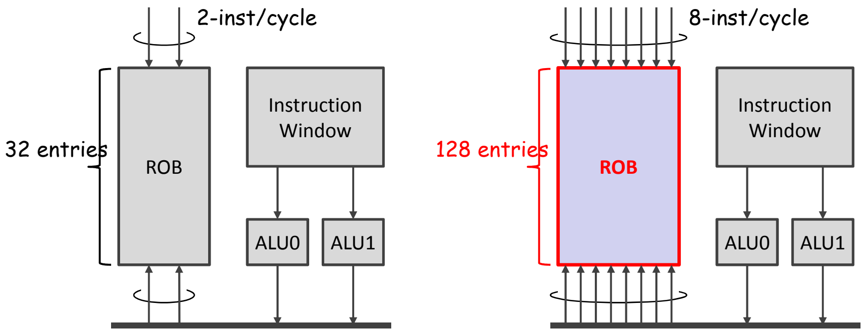
Comparison of Algorithms

- Leaf-node Steering achieves reduction of inter-core communication and alleviation of load imbalance at the same time
- But, it increases instruction count

Steering algorithms	 I0 I1 I2 I3 I4 I5 I6 Core0 Core1 Core2 Core3 (a) Modulo-2	 (b) Dependency-based	 I0 I1 I2 I0 I1 I3 I4 I5 I6 Core0 Core1 Core2 (c) Leaf-node
Inter-core communication overhead	×	○	○
Load imbalance overhead	○	×	△

Problems of Re-order Buffer on CoreSymphony [1/2]

- Reorder buffer (ROB) is a key structure of out-of-order processor to allow instructions to be retired in-order.
- Each ROB manages all instruction in all fused cores
  - Needs 4x ROB entries, compared to a 2-way out-of-order core

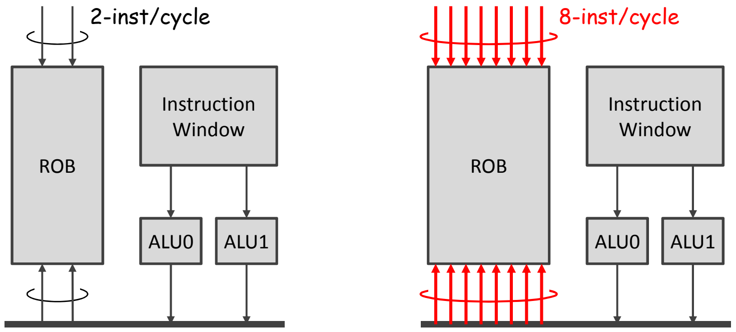


(a) Typical 2-way out-of-order core

(b) Conventional CoreSymphony

Problems of Re-order Buffer on CoreSymphony [2/2]

- On 4-core fusion, up to eight instructions are dispatched and completed for every one cycle
  - Needs 4x ROB ports, compared to a typical 2-way out-of-order core



(a) Typical 2-way out-of-order core

(b) Conventional CoreSymphony

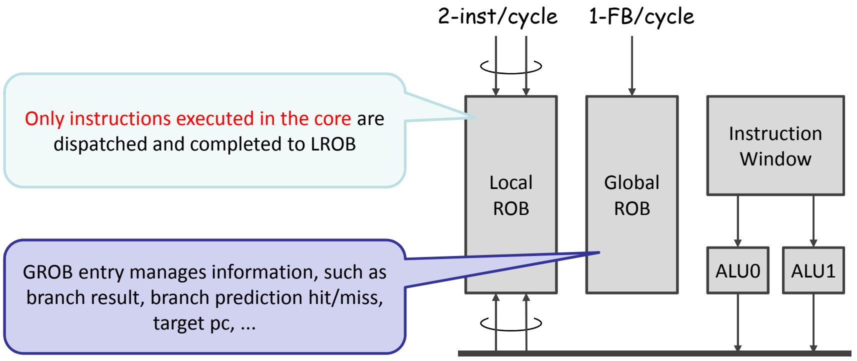
Approach to reduce ROB complexity

- Reduce the number of **ROB entries** and **ROB ports** to the complexity of 2-way out-of-order core
  - Conventional ROB:
    - Each ROB manages all instruction in all fused cores
  - Proposed ROB:
    - Each (Local) ROB manages only instructions executed in each core

- Share limited **necessary information** for precise control
  - Branch result (taken / not taken)
  - Branch prediction hit/miss
  - Branch target pc
  - ...

Proposal: Distributed Re-order Buffer

- Divide a large ROB into two small ROBs
  - Local Re-order Buffer (LROB)
  - Global Re-order Buffer (GROB)



Only instructions executed in the core are dispatched and completed to LROB

GROB entry manages information, such as branch result, branch prediction hit/miss, target pc, ...

(c) Realistic CoreSymphony

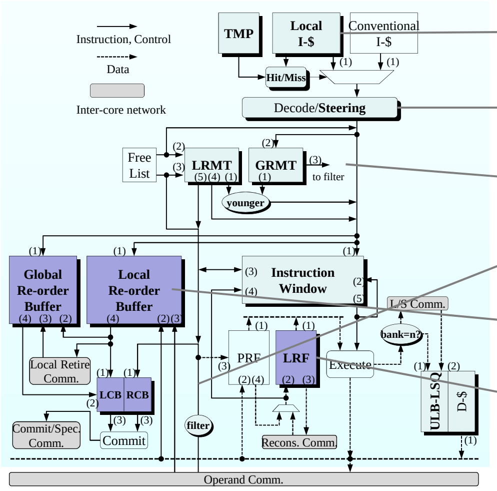
Comparison of ROB Hardware Complexity

Module	function	2-way out-of-order	8-way out-of-order	CoreSymphony (conventional)	CoreSymphony (proposal)
Local Re-order Buffer (LROB)	Allocate Update exec-flag (Local) Update exec-flag (Remote) Release	2 entry/cycle 2 Write - 2 entry/cycle	8 entry/cycle 8 Write - 8 entry/cycle	8 entry/cycle 2 Write 6 Write 8 entry/cycle	2 entry/cycle 2 Write 3 Write 2 entry/cycle
Global Re-order Buffer (GROB)	Allocate Update exec-flag (Local) Update exec-flag (Remote) Release	- - - -	- - - -	- - - -	1 entry/cycle 1 Write 3 Write 1 entry/cycle

- Distributed Re-order Buffer
  - Reduces the number of allocate/update/release ports
  - However, additional module (Global ROB) is needed

Overall Structure of CoreSymphony (1-core)

- ▭ : Added/modified modules on **conventional** CoreSymphony
- ▬ : Added/modified modules for **newly proposed (Realistic)** CoreSymphony



- Instruction Cache
  - Local Instruction Cache
- Instruction Steering
  - Leaf-node Steering
- Register Renaming
  - 2-way Renaming
- Physical Register File (PRF)
  - PRF Distribution
- Re-order Buffer (ROB)
  - ROB Distribution
- In-order State Management
  - In-order State Distribution

Implementation of CoreSymphony on FPGA

- Implement CoreSymphony in Verilog HDL
  - Number of lines: about 20,000 lines
- Run and tested CoreSymphony on FPGA
  - Evaluation Board
    - Xilinx Virtex-7 FPGA VC707 Evaluation Kit
    - Implemented 4 cores in the FPGA
    - Use block RAM in the FPGA as the main memory
  - Test Program
    - Control lighting-up LEDs on board by memory mapped I/O
    - Light-up LEDs by rotation
  - Frequency
    - 20MHz

Xilinx Virtex-7 FPGA VC707
Evaluation Kit
(http://www.xilinx.com)

We see that the performance is improved by increasing the number of fused cores

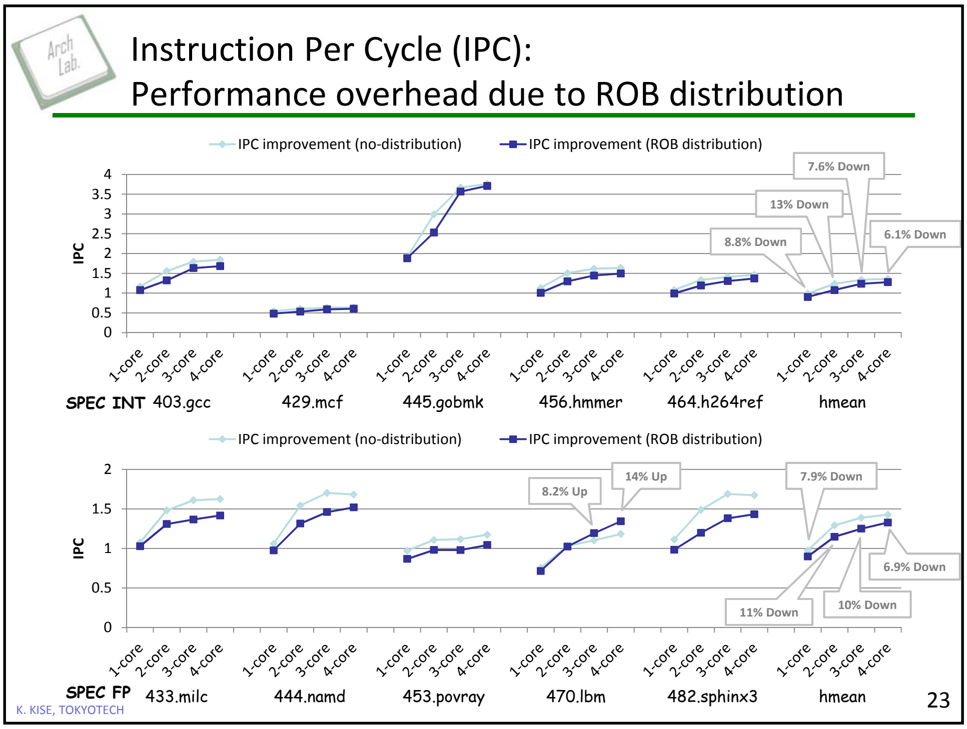
Evaluation Environment

- Software Simulation for performance evaluation
  - Use a cycle-level software-based simulator, SimMips[2]
  - Modify SimMips for the evaluation of CoreSymphony
  - Evaluate instruction per cycle (IPC)
  - Benchmarks
    - 5 INTs and 5 FPs from SPEC2006
      - 403.gcc, 429.mcf, 445.gobmk, 456.hmmer, 464.h264ref
      - 433.milc, 444.namd, 453.povray, 470.lbm, 482.sphinx3
    - Data set: test
- Synthesis Tool for hardware complexity
  - Xilinx ISE 14.3
  - Target Device: VC707
  - Target FPGA: xc7vx485t

Table: Baseline processor configuration

Functional Units	2 iALU, 1 LD/ST, 1 fpALU
Instruction Window	32/32 ent. for INT/FP
Local Re-order Buffer	32 entries
Global Re-order Buffer	8 entries
Physical Register File	64/64 entries for INT/FP
Load/Store Queue	128 entries ULB-LSQ
Memory disambiguation	1K entries LWT
Branch prediction	6 bit GHR, 2K entries TMP
Branch Target Buffer	1K entries, 2-way
L1-D$	16KB, 2-way, 1 cycle
L1-I$ (Conventional)	8KB, 2-way, 1 cycle
L1-I$ (Local)	8KB, 4-way, 2 cycle
Shared L2-$	2MB, 4-way, 10 cycle
Main memory latency	100 cycles
Inter-core latency	1 cycle

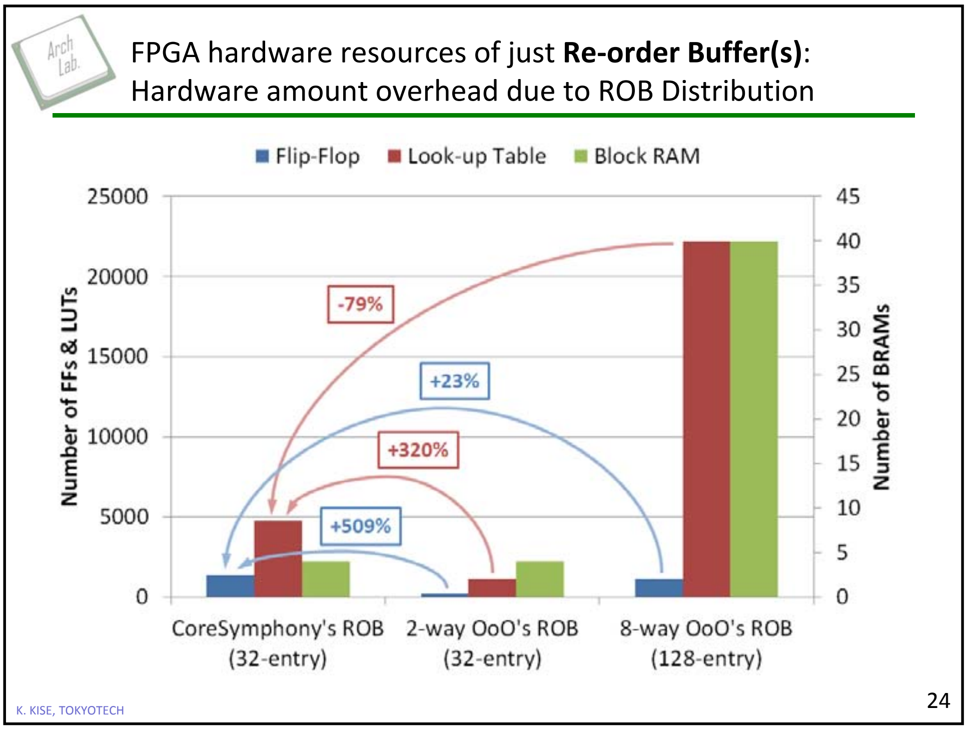
[2] N. Fujieda, et al. A MIPS System Simulator SimMips for Education and Research of Computer Science. *IPSJ Journal, Vol.50, No.11*, pp.2665-2676, 2009.



Instruction Per Cycle (IPC): Performance overhead due to ROB distribution

K. KISE, TOKYOTECH

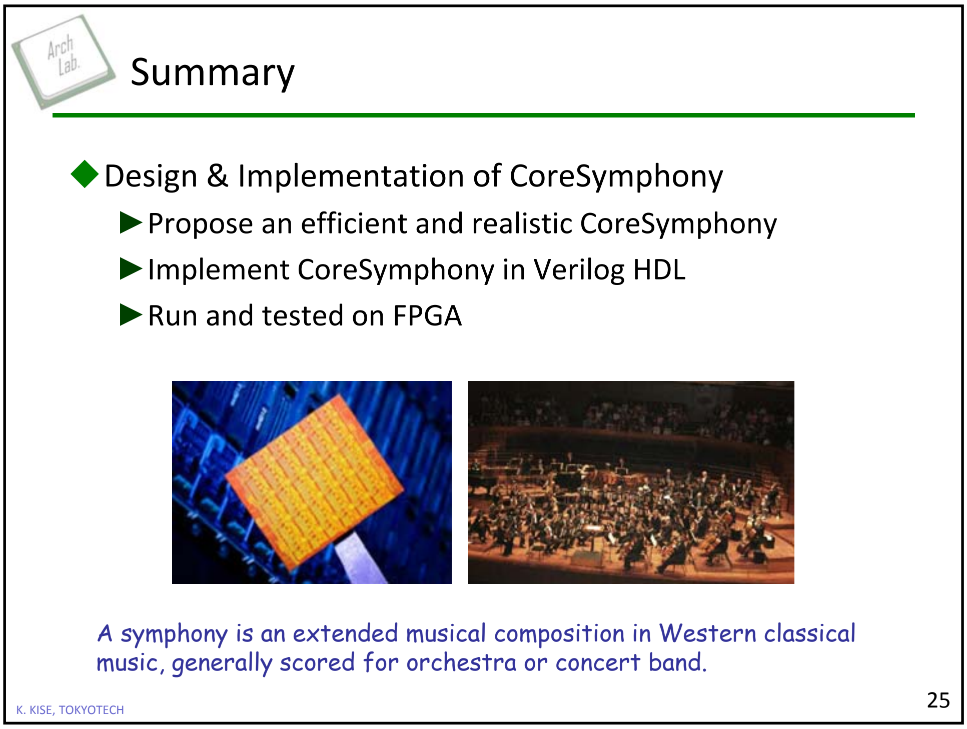
23



FPGA hardware resources of just **Re-order Buffer(s)**: Hardware amount overhead due to ROB Distribution

K. KISE, TOKYOTECH

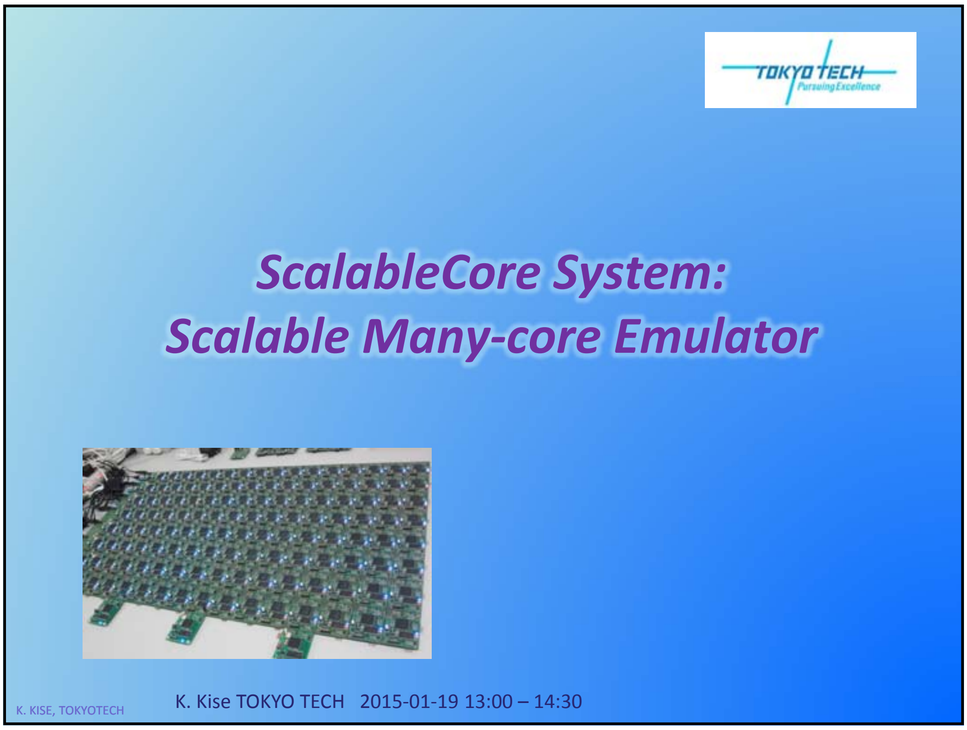
24



Summary

- Design & Implementation of CoreSymphony
  - Propose an efficient and realistic CoreSymphony
  - Implement CoreSymphony in Verilog HDL
  - Run and tested on FPGA

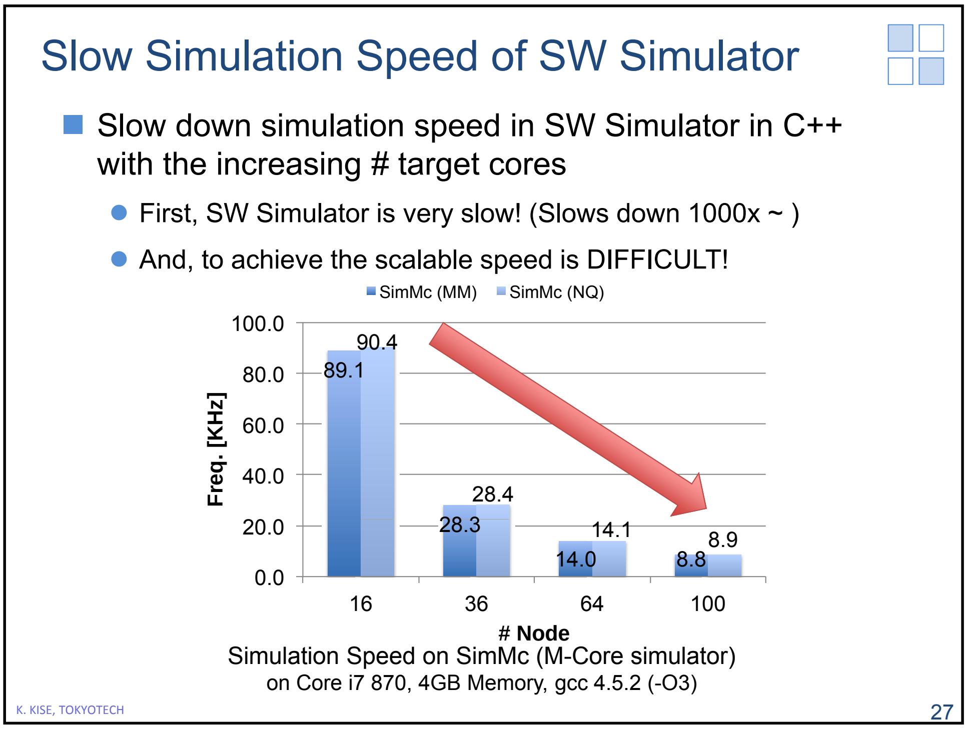
A symphony is an extended musical composition in Western classical music, generally scored for orchestra or concert band.



TOKYO TECH
Pursuing Excellence

ScalableCore System:
Scalable Many-core Emulator

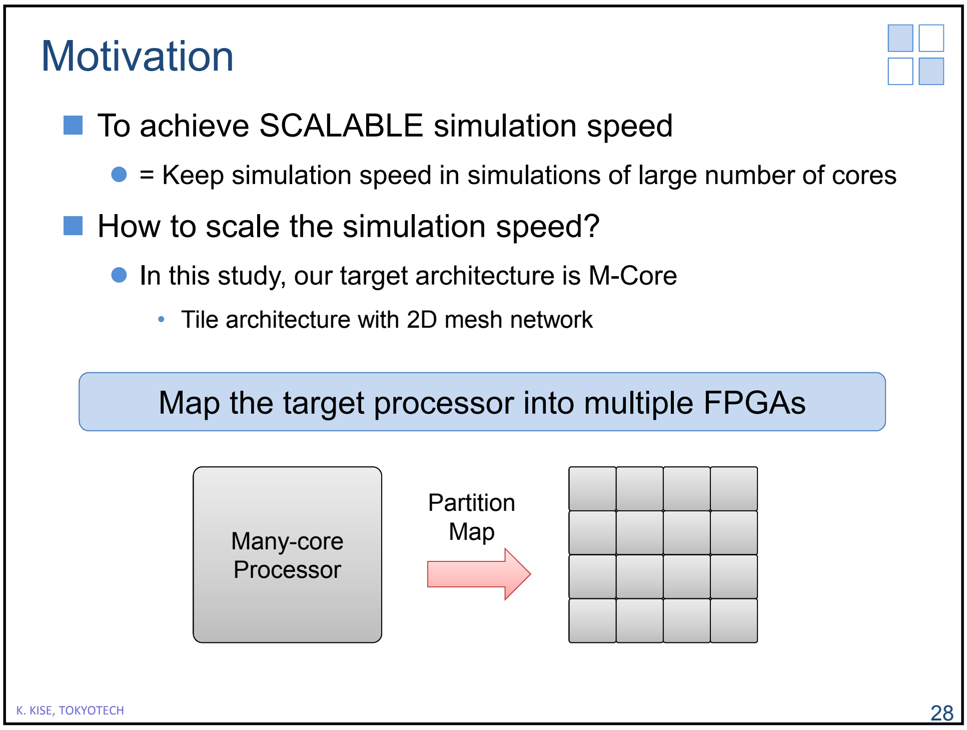
K. Kise TOKYO TECH 2015-01-19 13:00 – 14:30



Slow Simulation Speed of SW Simulator

- Slow down simulation speed in SW Simulator in C++ with the increasing # target cores
  - First, SW Simulator is very slow! (Slows down 1000x ~)
  - And, to achieve the scalable speed is DIFFICULT!

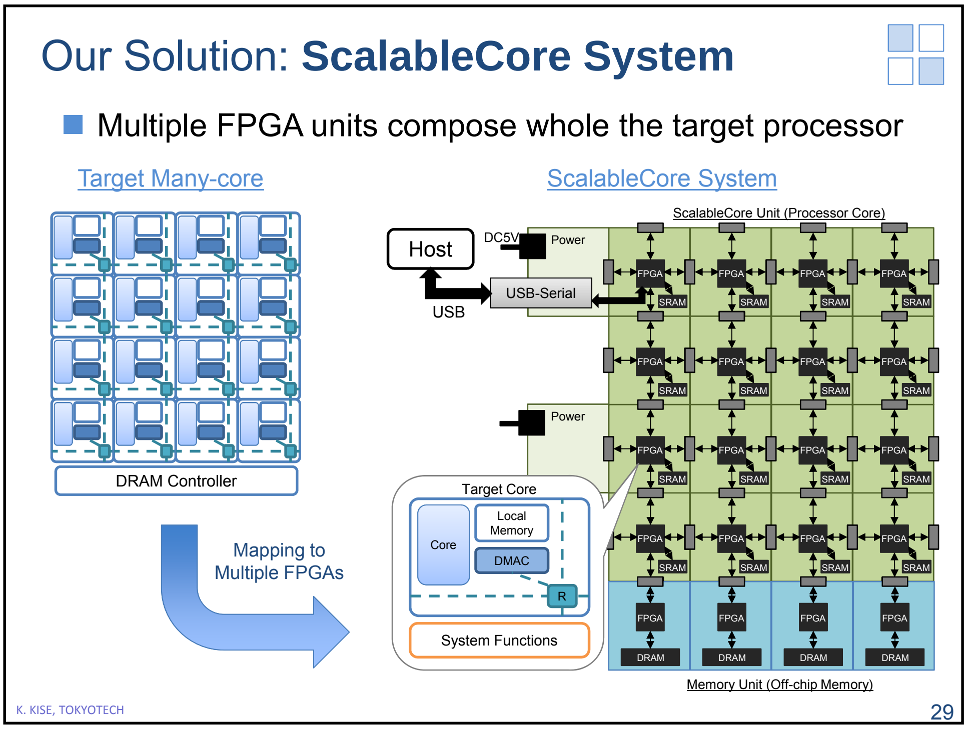
Simulation Speed on SimMc (M-Core simulator)
on Core i7 870, 4GB Memory, gcc 4.5.2 (-O3)



Motivation

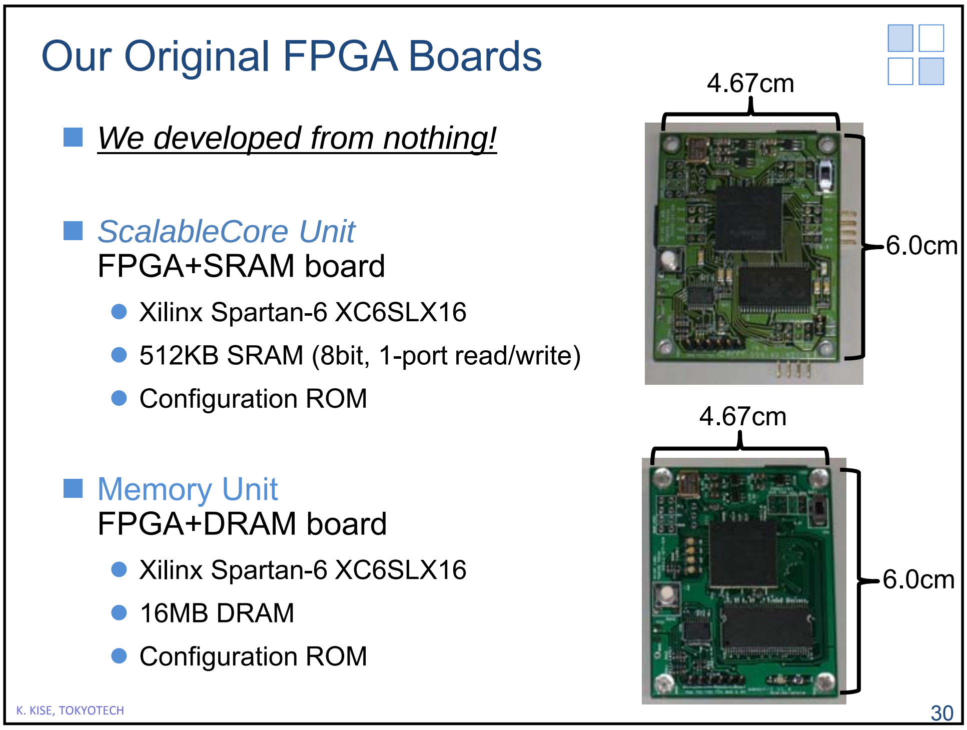
- To achieve SCALABLE simulation speed
  - = Keep simulation speed in simulations of large number of cores
- How to scale the simulation speed?
  - In this study, our target architecture is M-Core
    - Tile architecture with 2D mesh network

Map the target processor into multiple FPGAs



Our Solution: **ScalableCore System**

- Multiple FPGA units compose whole the target processor



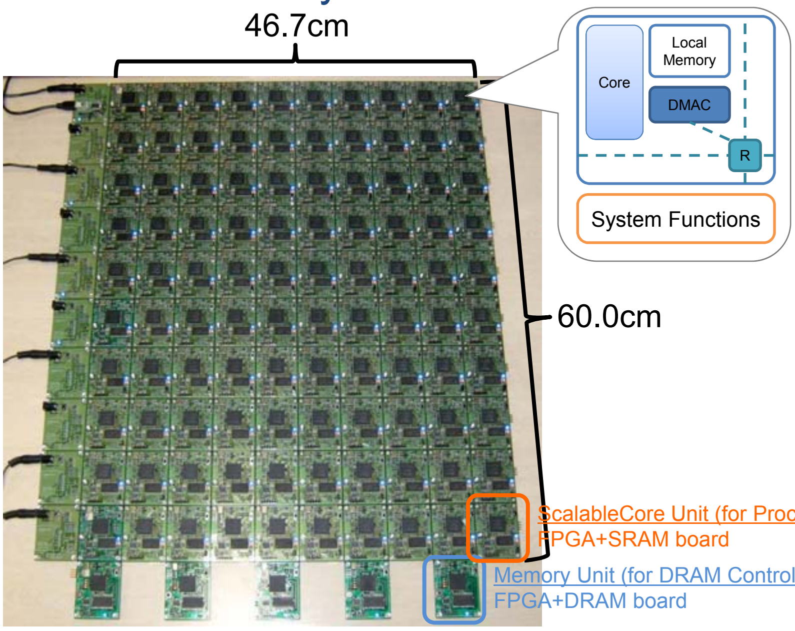
Our Original FPGA Boards

- ***We developed from nothing!***

- *ScalableCore Unit*
  FPGA+SRAM board
  - Xilinx Spartan-6 XC6SLX16
  - 512KB SRAM (8bit, 1-port read/write)
  - Configuration ROM

- Memory Unit
  FPGA+DRAM board
  - Xilinx Spartan-6 XC6SLX16
  - 16MB DRAM
  - Configuration ROM

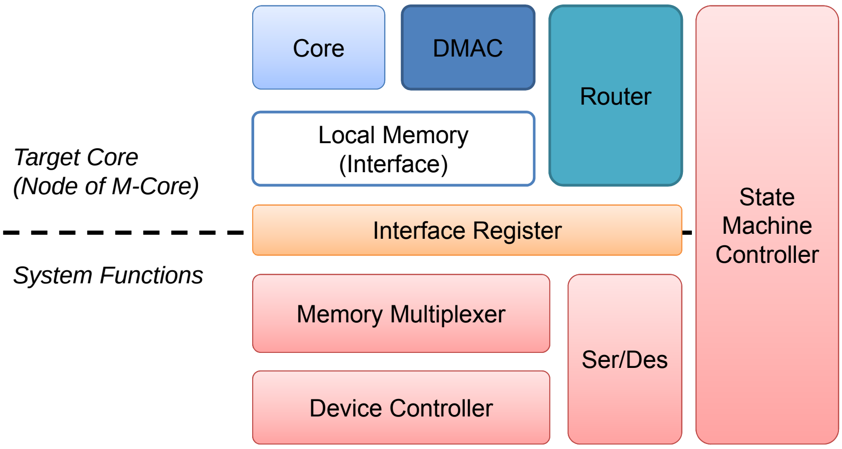
ScalableCore System 3.3 for 100-Nodes



ScalableCore Unit (for Processor Core): FPGA+SRAM board

Memory Unit (for DRAM Controller): FPGA+DRAM board

Logic Hierarchy of ScalableCore Unit



Target Core (Node of M-Core)

System Functions

Core

DMAC

Router

Local Memory (Interface)

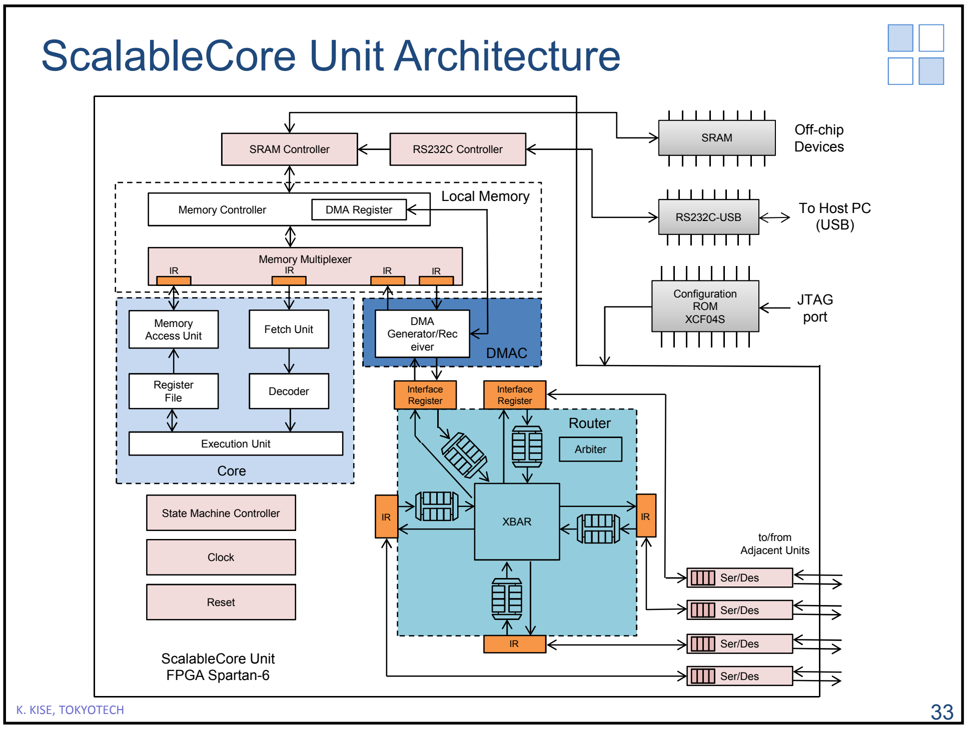
Interface Register

Memory Multiplexer

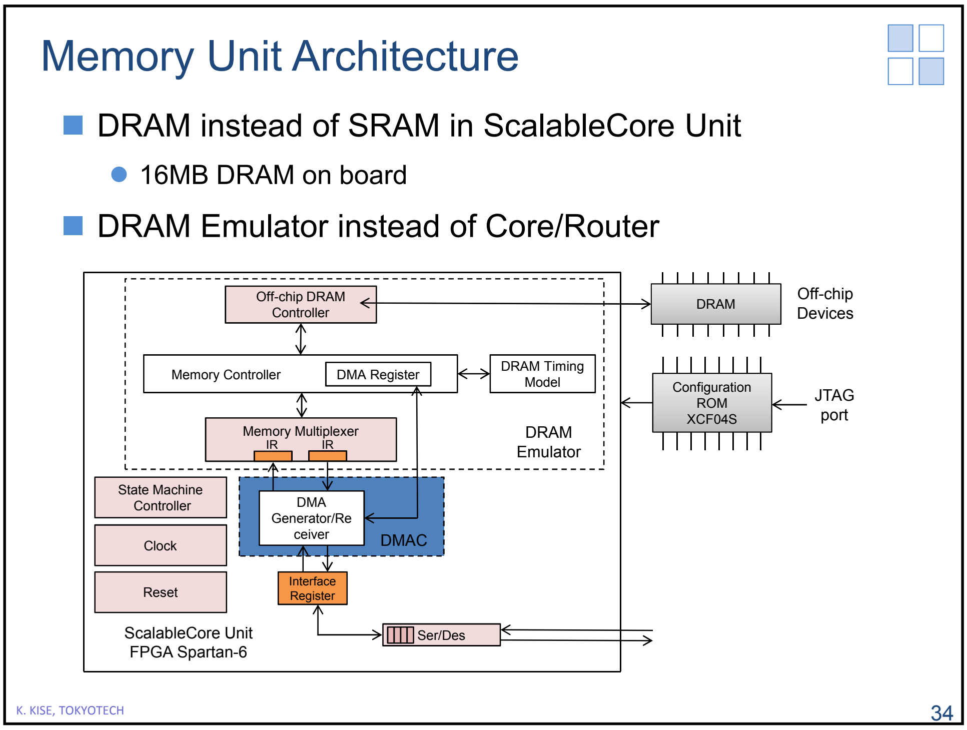
Device Controller

Ser/Des

State Machine Controller

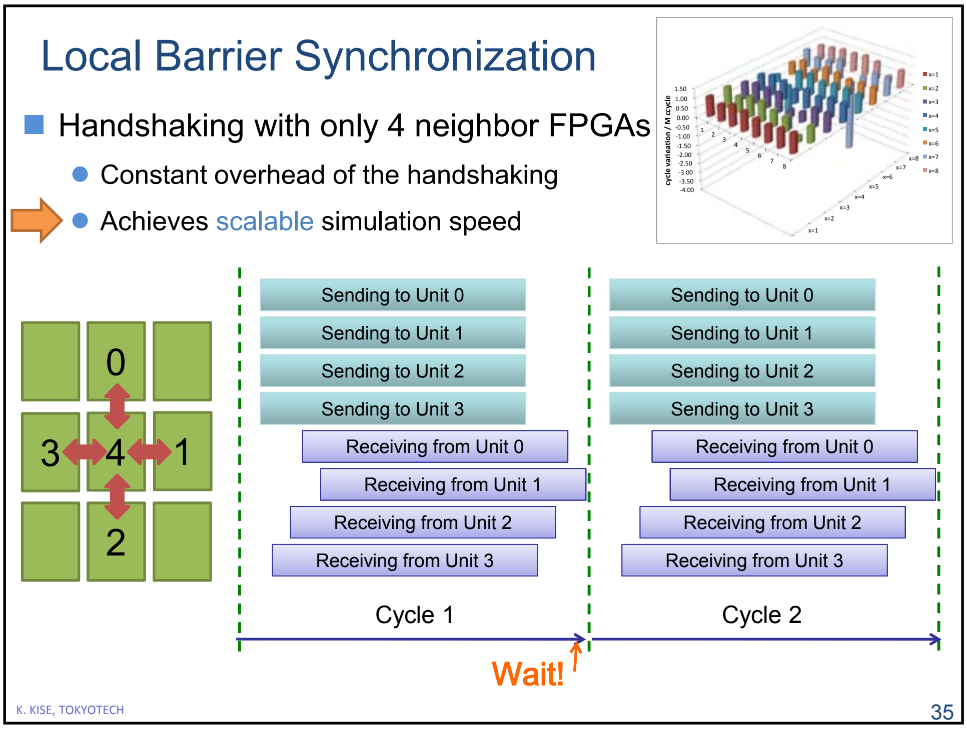


ScalableCore Unit Architecture



Memory Unit Architecture

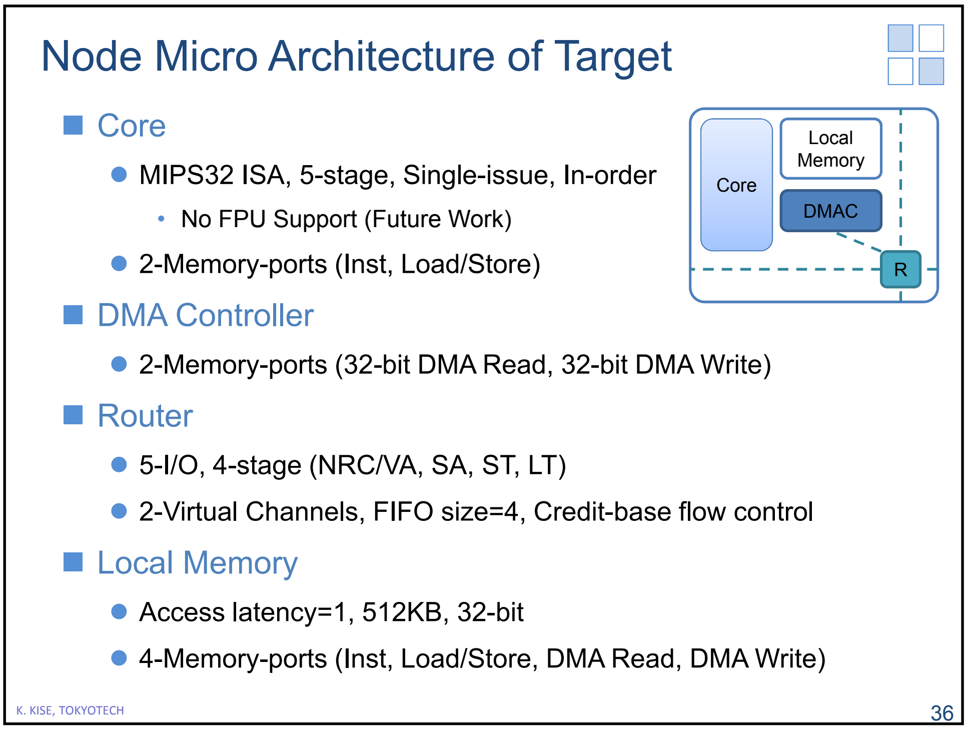
- DRAM instead of SRAM in ScalableCore Unit
  - 16MB DRAM on board
- DRAM Emulator instead of Core/Router



Local Barrier Synchronization

- Handshaking with only 4 neighbor FPGAs
  - Constant overhead of the handshaking
  - Achieves scalable simulation speed

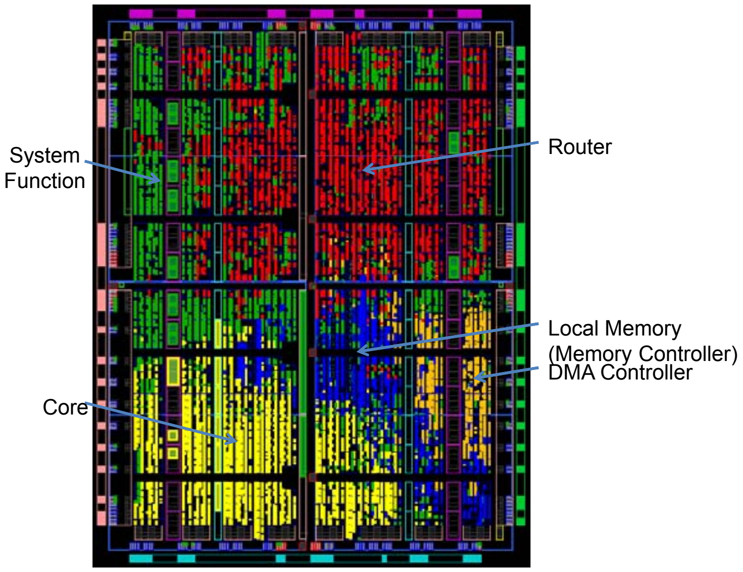
0
3
4
1
2
Sending to Unit 0
Sending to Unit 1
Sending to Unit 2
Sending to Unit 3
Receiving from Unit 0
Receiving from Unit 1
Receiving from Unit 2
Receiving from Unit 3
Cycle 1
Sending to Unit 0
Sending to Unit 1
Sending to Unit 2
Sending to Unit 3
Receiving from Unit 0
Receiving from Unit 1
Receiving from Unit 2
Receiving from Unit 3
Cycle 2
Wait!
cycle variation / M cycle



Node Micro Architecture of Target

- Core
  - MIPS32 ISA, 5-stage, Single-issue, In-order
    - No FPU Support (Future Work)
  - 2-Memory-ports (Inst, Load/Store)
- DMA Controller
  - 2-Memory-ports (32-bit DMA Read, 32-bit DMA Write)
- Router
  - 5-I/O, 4-stage (NRC/VA, SA, ST, LT)
  - 2-Virtual Channels, FIFO size=4, Credit-base flow control
- Local Memory
  - Access latency=1, 512KB, 32-bit
  - 4-Memory-ports (Inst, Load/Store, DMA Read, DMA Write)

ScalableCore Unit Floorplan (XC6SLX16)



Resource Utilization of ScalableCore Unit

- FPGA: Spartan-6 XC6SLX16
- NOT serious resource utilization by system function
  - System: 20% LUTs and 15% Regs (of LX16)
  - Target: 64% LUTs and 14% Regs (of LX16)

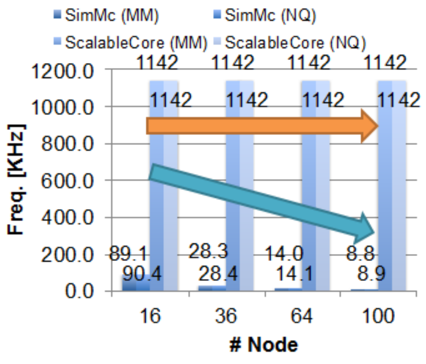
Module	LUT	Register	BRAM	LUTRAM	DSP
System Function	1700	2693	16	0	0
Core	1920	713	3	0	6
DMA Controller	444	378	0	0	0
DMA Register	590	535	0	0	0
Router	2475	959	0	280	0
Target Total	5429	2585	3	280	6
Total	7129	5278	19	280	6
Percent Utilization	84%	29%	31%	N/A	6%

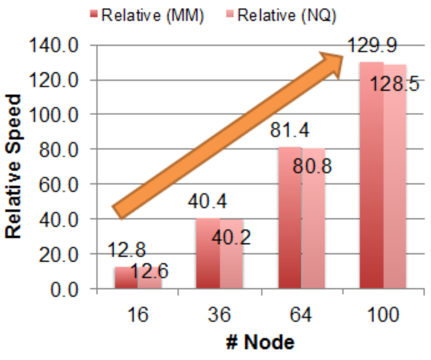
Simulation Speed

- Environment
  - ScalableCore system 3.3 (FPGA-based simulator of M-Core)
    - Freq.: 40MHz (SerDes: 80MHz)
  - SimMc (Software simulator of M-Core)
    - Intel Corei7 870, Memory 4GB, gcc4.5.2 (-O3), Ubuntu Server 11.04
- # Node
  - 16 (4x4), 36 (6x), 64 (8x8), 100 (10x10)

Simulation Speed [KHz]

- ScalableCore System achieves constant simulation frequency: Good weak-scaling
- With # target core increases, relative speed increases!!
  - In 100-Node, ScalableCore system runs at 129x faster



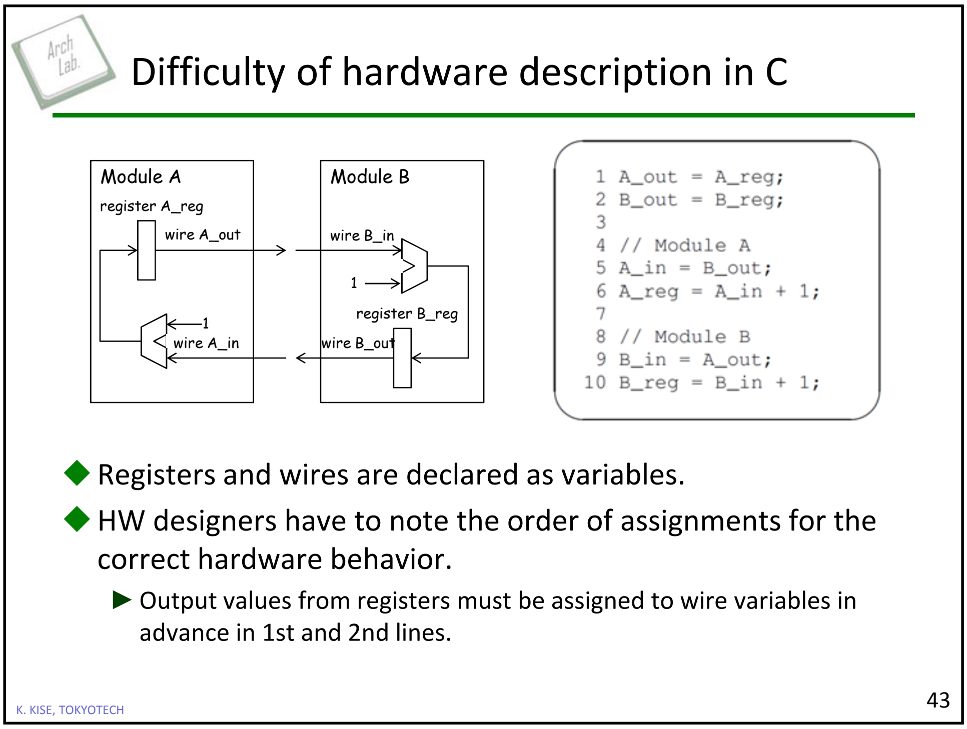


ArchHDL: A novel Hardware RTL Design Environment in C++

K. Kise TOKYO TECH 2015-01-19 13:00 – 14:30

Motivation

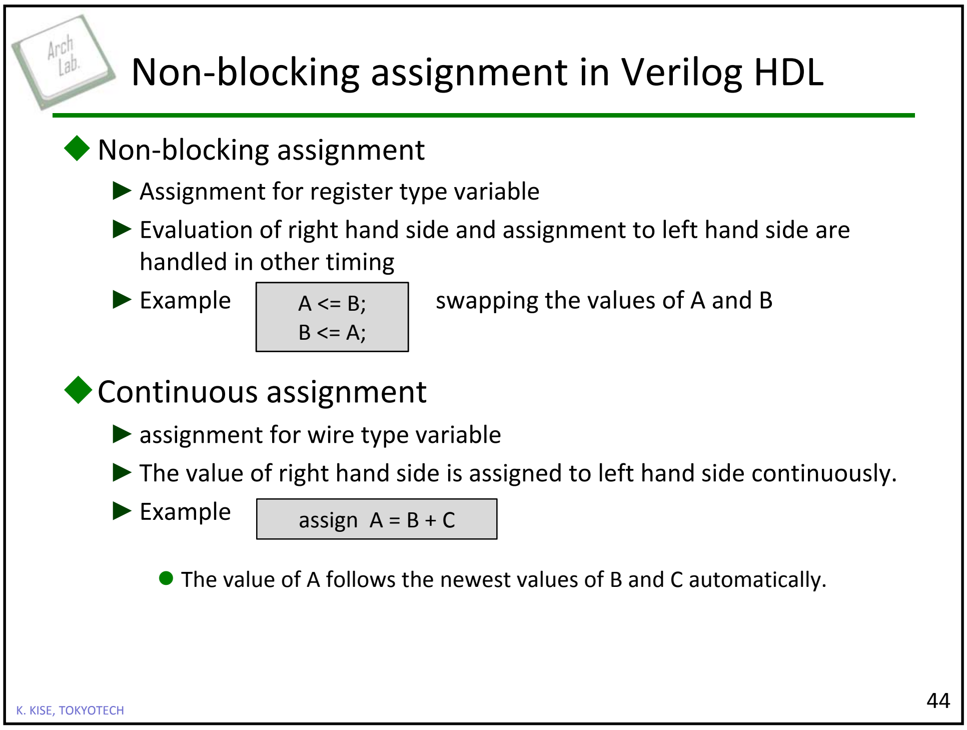
- Verilog simulation for architectural/logic design verification is very slow!
- The simulation does not support thread-level parallelism and many-core.
- Sometimes, Verilog HDL is not familiar and C/C++ environment is good for coding and debugging.
- We designed **ArchHDL**, a new hardware description language for high-speed HW architectural evaluations.
  - C++ based.
  - Parallel simulation/many-core friendly by birth.



Difficulty of hardware description in C

```
 1 A_out = A_reg;
 2 B_out = B_reg;
 3
 4 // Module A
 5 A_in = B_out;
 6 A_reg = A_in + 1;
 7
 8 // Module B
 9 B_in = A_out;
10 B_reg = B_in + 1;
```

- Registers and wires are declared as variables.
- HW designers have to note the order of assignments for the correct hardware behavior.
  - Output values from registers must be assigned to wire variables in advance in 1st and 2nd lines.



Non-blocking assignment in Verilog HDL

- Non-blocking assignment
  - Assignment for register type variable
  - Evaluation of right hand side and assignment to left hand side are handled in other timing
  - Example

```
A <= B;
B <= A;
```

swapping the values of A and B

- Continuous assignment
  - assignment for wire type variable
  - The value of right hand side is assigned to left hand side continuously.
  - Example

```
assign  A = B + C
```

- The value of A follows the newest values of B and C automatically.

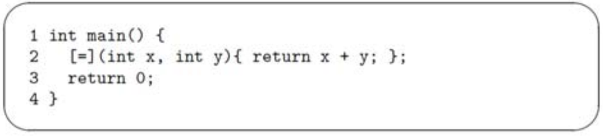
ArchHDL, new language for RTL modeling

- Non-blocking assignment and continuous assignment support in C++
  - Verilog HDL like hardware description
  - **The key is to use lambda function of C++11**
- High-speed simulation
  - Parallel simulation/many-core friendly by birth

- **Current limitations**
  - support only one clock signal
  - The assignment timing to registers is only in the positive edge clock
  - support only C++ data types and operators
    - The arbitrary bit length of data types are not supported
    - Bit selection and bit concatenation are also not supported

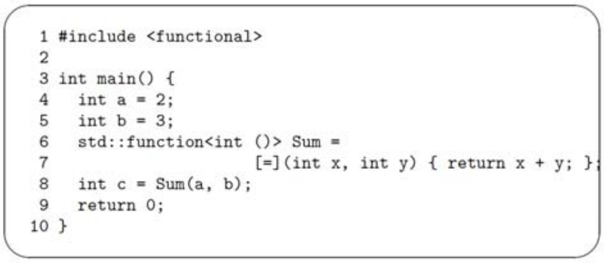
Lambda function of C++11

- The 2nd line is the only definition of a lambda function (anonymous function)



```
1 int main() {
2   [=](int x, int y){ return x + y; };
3   return 0;
4 }
```

- Sum becomes a functional object which takes two integer value as argument and returns sum of them.



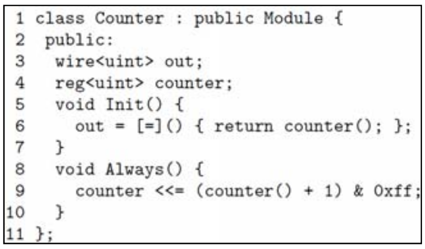
```
#include <functional>

int main() {
  int a = 2;
  int b = 3;
  std::function<int ()> Sum =
                    [=](int x, int y) { return x + y; };
  int c = Sum(a, b);
  return 0;
}
```

8-bit counter in ArchHDL and Verilog HDL

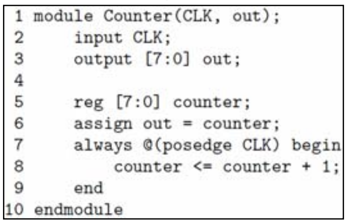
- Description in ArchHDL is similar to that in Verilog HDL.
  - **Module**, **wire** and **reg** classes are defined in ArchHDL library
  - **Init** or **Assign** and **Always** functions also defined in ArchHDL library
  - Redefine the “**<<=**“ operator for non-blocking assignment
- The major differences are related to the clock signal.

ArchHDL



```
class Counter : public Module {
 public:
  wire<uint> out;
  reg<uint> counter;
  void Init() {
    out = [=]() { return counter(); };
  }
  void Always() {
    counter <<= (counter() + 1) & 0xff;
  }
};
```

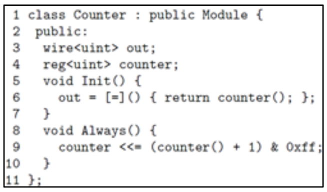
Verilog HDL



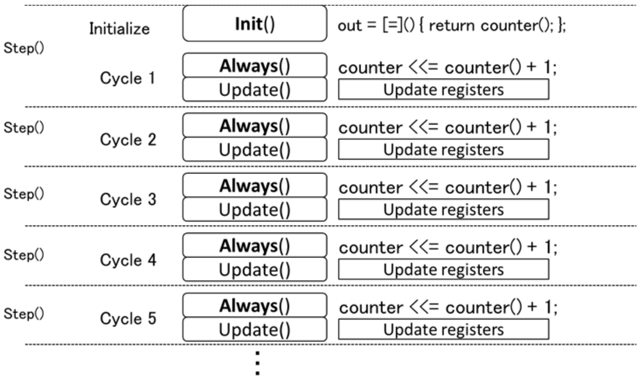
```
module Counter(CLK, out);
    input CLK;
    output [7:0] out;

    reg [7:0] counter;
    assign out = counter;
    always @(posedge CLK) begin
        counter <= counter + 1;
    end
endmodule
```

Simulation process of ArchHDL



```
class Counter : public Module {
 public:
  wire<uint> out;
  reg<uint> counter;
  void Init() {
    out = [=]() { return counter(); };
  }
  void Always() {
    counter <<= (counter() + 1) & 0xff;
  }
};
```



Step() Initialize **Init()** out = [=]() { return counter(); };

Cycle 1 **Always()** counter <<= counter() + 1;
Update() Update registers

Step() Cycle 2 **Always()** counter <<= counter() + 1;
Update() Update registers

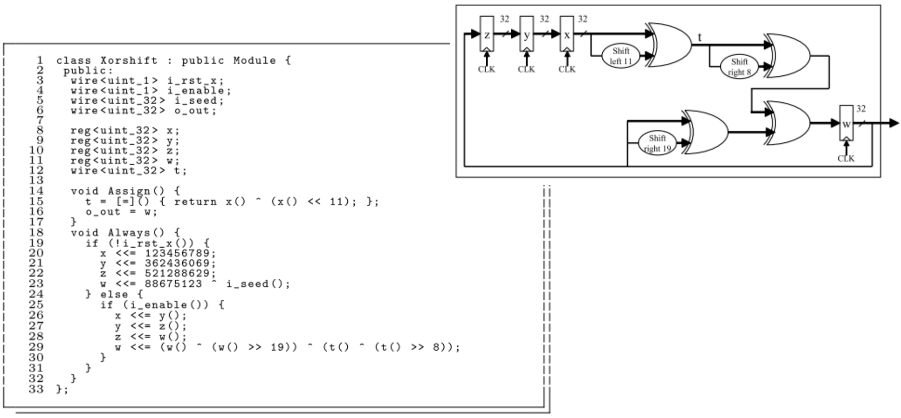
Step() Cycle 3 **Always()** counter <<= counter() + 1;
Update() Update registers

Step() Cycle 4 **Always()** counter <<= counter() + 1;
Update() Update registers

Step() Cycle 5 **Always()** counter <<= counter() + 1;
Update() Update registers

⋮

Xorshift random generator in ArchHDL



```
class Xorshift : public Module {
 public:
  wire<uint_1> i_rst_x;
  wire<uint_1> i_enable;
  wire<uint_32> i_seed;
  wire<uint_32> o_out;

  reg<uint_32> x;
  reg<uint_32> y;
  reg<uint_32> z;
  reg<uint_32> w;
  wire<uint_32> t;

  void Assign() {
    t = [=]() { return x() ^ (x() << 11); };
    o_out = w;
  }
  void Always() {
    if (!i_rst_x()) {
      x <<= 123456789;
      y <<= 362436069;
      z <<= 521288629;
      w <<= 88675123 ^ i_seed();
    } else {
      if (i_enable()) {
        x <<= y();
        y <<= z();
        z <<= w();
        w <<= (w() ^ (w() >> 19)) ^ (t() ^ (t() >> 8));
      }
    }
  }
};
```

Fig. 2. A sample description of Xorshift random value generator in ArchHDL.

Simulation speed evaluation

- Many-core processor employing scratch-pad memories
  - The PE is a 32-bit five-stage pipelined MIPS processor
  - The router architecture is the conventional Input-buffered Virtual Channel router with five-stages pipeline

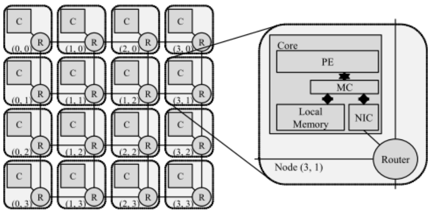


Fig. 5. M-Core Architecture used as a practical hardware for evaluation.

- CPU: Intel Xeon E5-2687W
- Memory: 32 GB
- OS: Ubuntu 13.04 x86_64
- GCC (g++): version 4.7.3, optimization -Ofast
- Intel Compiler (ICPC): version 14.0.1 optimization -Ofast
- Synopsys VCS: version H-2013.06

Evaluation result

- Simulation time varying the number of nodes on many-core
- The simulation speed of ArchHDL was about **4.5x** faster than the simulation speed using Synopsys VCS which is one of the fastest Verilog HDL simulator (and an expensive commercial product).

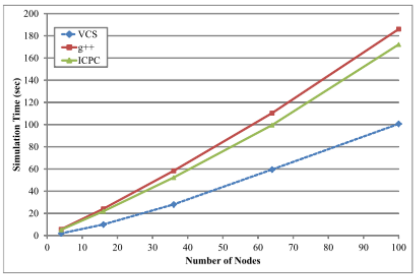
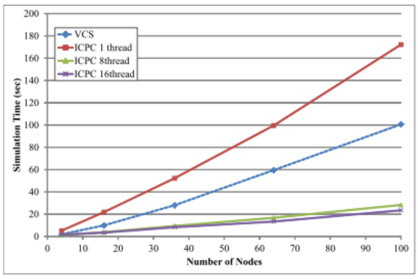


Fig. 6. The simulation time comparison using just single-threaded.



The parallelized simulation time in ArchHDL. Note that the simulation time of VCS is single-thread.

High-speed FPGA Accelerator for Integer Sorting

K. Kise TOKYO TECH 2015-01-19 13:00 – 14:30

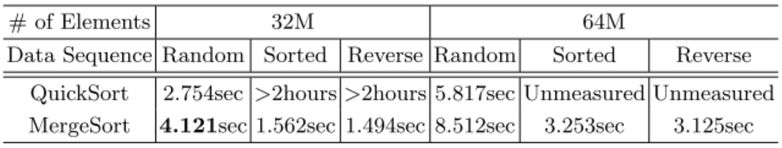
Motivation

- Is an FPGA sorting at 200MHz faster than a software running on PC at 3.4GHz ?
  - Frequency slowdown of FPGA is **17**!
- If the FPGA sorting is faster than PC, how fast it is?
  - Is it effective if it achieves **1.2x** speedup in the real world ?
  - Is it effective if it achieves **2x** speedup in the real world ?
  - Is it effective if it achieves **4x** speedup in the real world ?
- Our target speedup is …

32-bit Integer Sorting

- Desktop computer
  - Intel Core i7, 3.4GHz
  - DDR3 16GB memory
  - Microsoft Visual C++ Compiler 2013, /Ox optimization
- Baseline
  - Merge sort
    32 mega elements, 4.12 seconds
    256 mega elements, 32.8 seconds

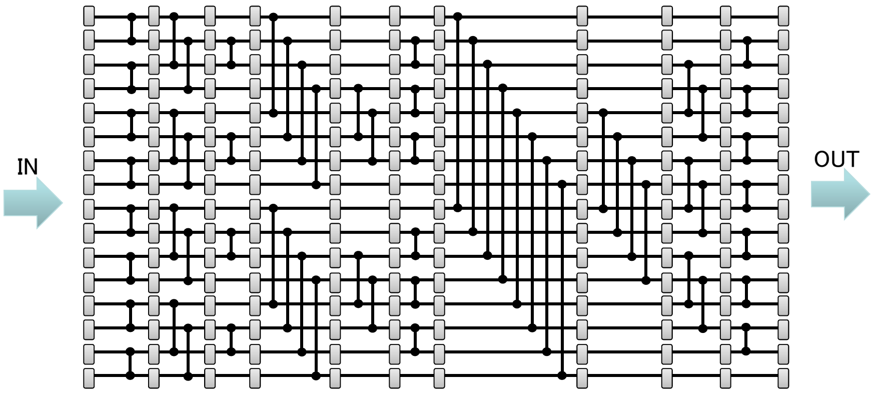
Table 1. Execution Time of Quick Sort and Merge Sort



# of Elements	32M			64M		
Data Sequence	Random	Sorted	Reverse	Random	Sorted	Reverse
QuickSort	2.754sec	>2hours	>2hours	5.817sec	Unmeasured	Unmeasured
MergeSort	**4.121**sec	1.562sec	1.494sec	8.512sec	3.253sec	3.125sec

Sorting Network

- Batcher's odd–even merge sort
- High throughput, 16 elements output / cycle

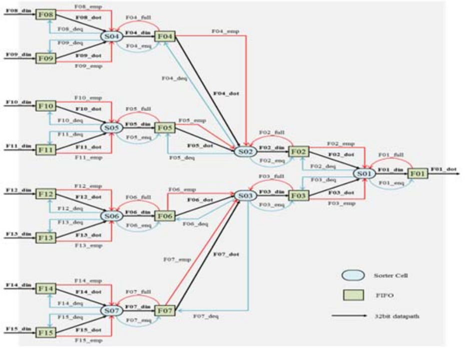


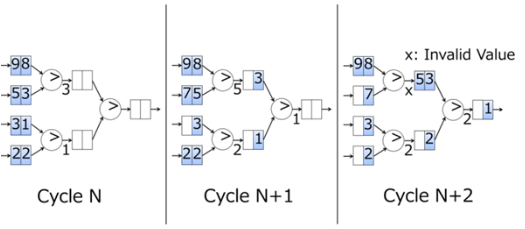
compare & swap values if top value is greater than bottom
32bit register

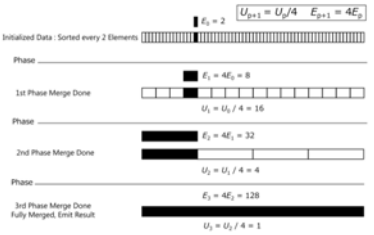
Merge Sorter Tree

5. Koch, D., Torresen, J.: Fpgasort: A high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting. In: Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays. pp. 45–54. FPGA '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1950413.1950427

- General sorting hardware
- Low throughput, 1 element output / cycle

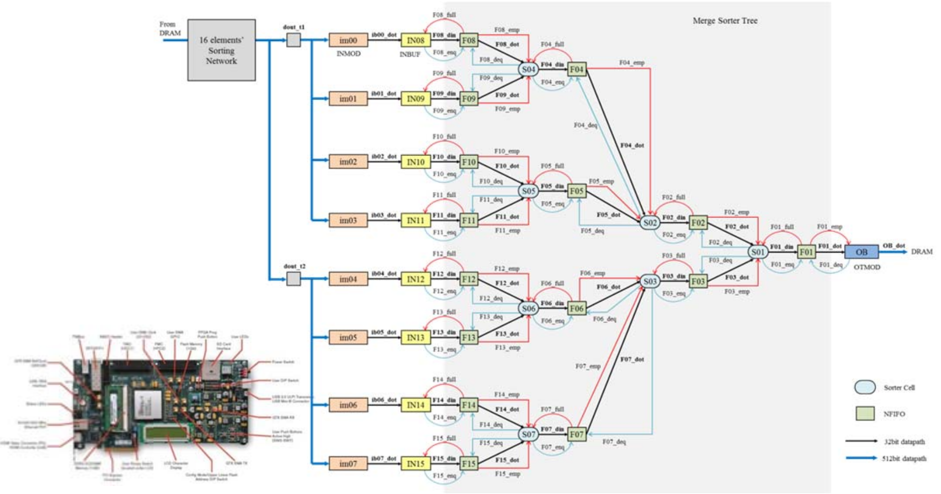






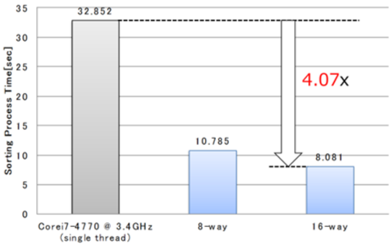
Our FPGA sorting solution

T. Usui, R. Kobayashi, and K. Kise: A Challenge of Portable and High-speed FPGA Accelerator, The 11th International Symposium on Applied Reconfigurable Computing (ARC2015) (April 2015).



Performance

- We implemented the proposal in Verilog HDL and verified it.
- Our FPGA sorting at 200MHz achieved **4x** faster than a software implementation.
  - ▶Frequency 200MHz is 17x slower than PC at 3.4GHz
    note: DRAM frequency for FPGA and PC is the same
  - ▶FPGA implementation is 68x efficient
  - ▶ So, total speedup of 68/17 = 4
- We have a feasible plan to enhance the speedup **over 10x**.
  - ▶multiple-outputs sorter tree
  - ▶parallel sorter trees



Fast and Accurate Emulation of Large-scale Network on Chip Architectures on a Single FPGA

K. Kise TOKYO TECH 2015-01-19 13:00 – 14:30

Motivation and ultra-speedup

- FPGA is the most suitable for hardware emulations.
  - FPGAs are often used for prototype systems for ASICs.
- In such fields, we show the ultra-speedup by an FPGA.

- Emulation of Large-scale (128x128 = 16,384 nodes) Network on Chip using just a single FPGA (DRAM is not used)
- We show that it can achieve **2,746x** speedup over Booksim (one of the most widely used software-based NoC simulator from Stanford Univ.)
  - Booksim takes 131 hours (5.5 days).
  - FPGA takes only 2.9 minutes.
- Note that it is cycle accurate!

VC707 FPGA Board

Simulation speed of Booksim

- The simulation speed of Booksim (single thread) decreases more than 17 times when the number of nodes is increased 9 times.
- It is hard to improve the simulation speed by using thread-level parallelism because of the huge amount of synchronizations between threads.

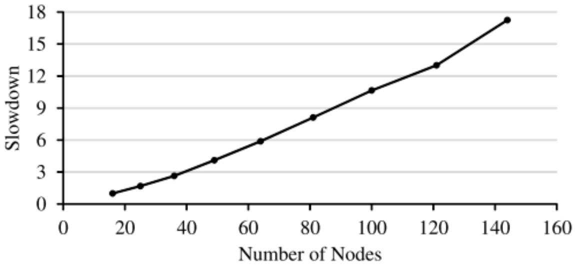
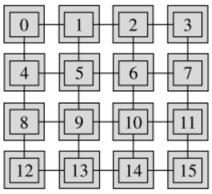


Figure 1. Slowdown in simulation speed of Booksim when changing the simulated NoC size from 16 nodes to 144 nodes.

Router architecture, node, mesh NoC

- Emulation model:
  typical mesh NoC of VC routers



4x4 mesh NoC (16 nodes)

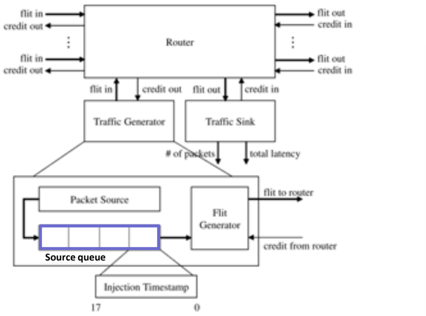


Figure 3. Architecture of each node in the proposed FPGA-based NoC emulator.

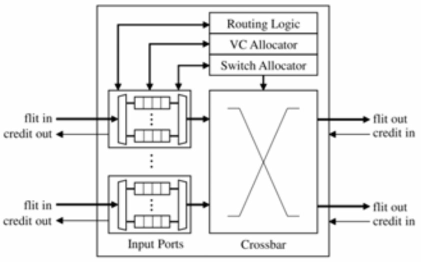


Figure 4. State-of-the-art VC router architecture.

Key idea (1)
Timeline of a packet source and the network

- The packet source is allowed to run behind the network.
- The state of the source queue is determined by both the network's time and the packet source's time.

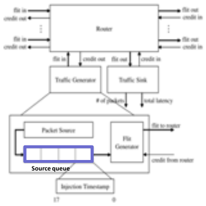
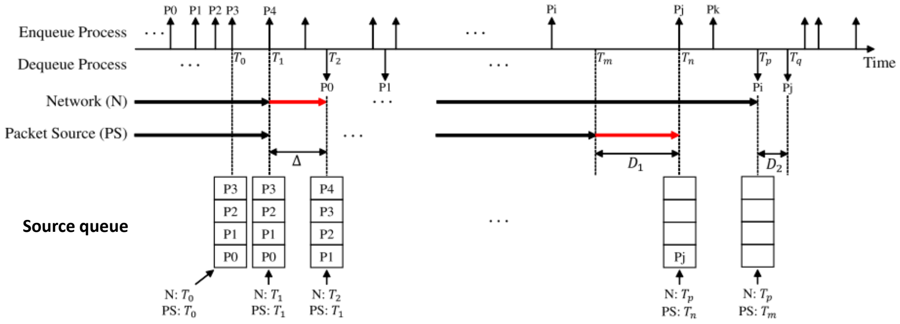


Figure 3. Architecture of each node in the proposed FPGA-based NoC emulator.



Key idea (2)
Time-Division Multiplexing (TDM)

- Not only by slice limitation, but also for memory ports reduction
- Although Virtex-7 VX485T has 1,030 BRAMs (total 4MB), remember that we are emulating 16,384 nodes!

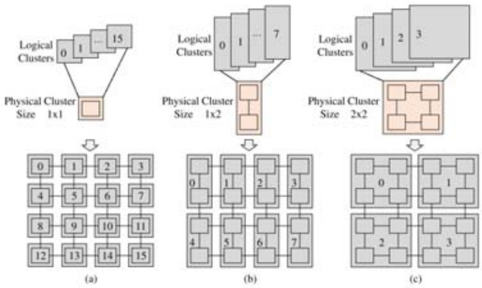
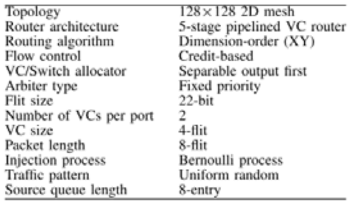
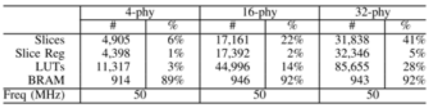


Table I
CONFIGURATION PARAMETERS



Topology	128×128 2D mesh
Router architecture	5-stage pipelined VC router
Routing algorithm	Dimension-order (XY)
Flow control	Credit-based
VC/Switch allocator	Separable output first
Arbiter type	Fixed priority
Flit size	22-bit
Number of VCs per port	2
VC size	4-flit
Packet length	8-flit
Injection process	Bernoulli process
Traffic pattern	Uniform random
Source queue length	8-entry

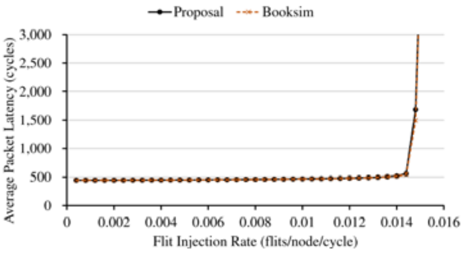
Table II
IMPLEMENTATION RESULTS IN THREE CASES: PHYSICAL CLUSTER SIZE = 2×2 (4-PHY), 4×4 (16-PHY), AND 8×4 (32-PHY).

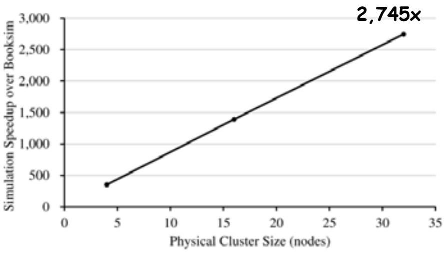


	4-phy		16-phy		32-phy	
	#	%	#	%	#	%
Slices	4,905	6%	17,161	22%	31,838	41%
Slice Reg	4,398	1%	17,392	2%	32,346	5%
LUTs	11,317	3%	44,996	14%	85,655	28%
BRAM	914	89%	946	92%	943	92%
Freq (MHz)	50		50		50	

Average packet latency and speedup

- Booksim takes 131 hours (5.5 days).
- FPGA emulator at 50MHz of 32 physical nodes takes only 2.9 minutes.
- There is a minor difference between two graphs when the traffic injection rate is high because we use a different pseudo-random number generator.
- Our FPGA-based NoC emulator can achieve **2,745x** simulation speedup over Booksim while using less than 41% of the total number of available slices of the Virtex-7 VX485T FPGA.





Agenda

- **Recent projects**
  - **CoreSymphony: Efficient and Realistic Cooperative Core Architecture**
  - **ScalableCore System: Scalable Many-core Emulator**
  - Challenges for Dependable Many-Core Processors
- **Ongoing projects**
  - **ArchHDL: A novel Hardware RTL Design Environment in C++**
  - **High-speed FPGA Accelerator for Integer Sorting**
  - **Fast and Accurate Emulation of Large-scale Network on Chip**
  - **MieruSys Project : Developing an Innovative Computer System**