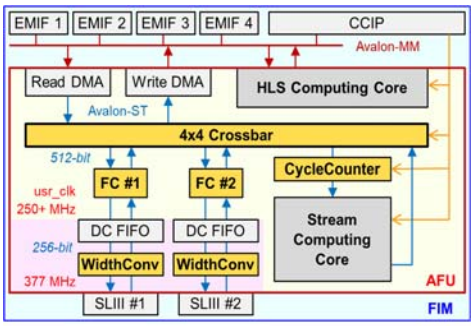
FPGAを用いた
カスタム高性能計算システムと
自動チューニングへの期待

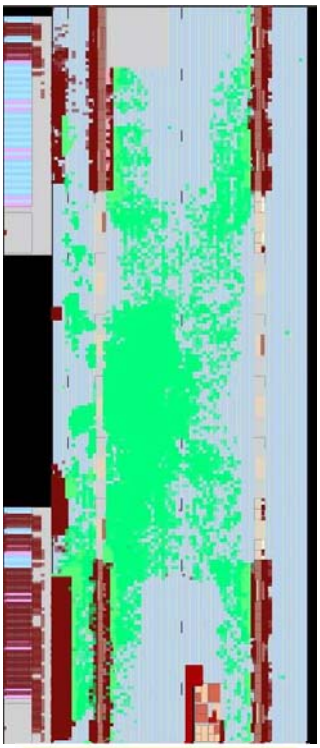
理化学研究所 計算科学研究センター

チームリーダー　佐野 健太郎

本講演

- 背景、目的、研究活動
- FPGAの近年の傾向
- 既存マシンに付加価値を
  与えるシステムプロジェクト
- FPGAクラスタ開発
- 自動チューニングへの期待
- 終わりに





Chip layout on FPGA

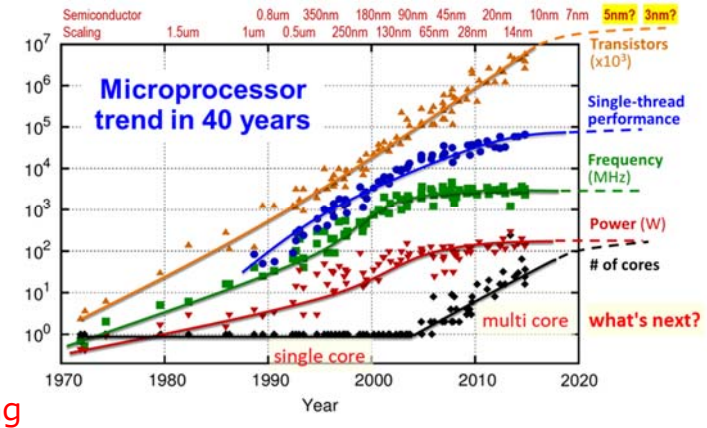
Processor Research Team

- **Goal**
  Explore and develop new processor/system architectures
  + suitable for next-generation HPC systems
  + with a novel computing/programming model
- **Targets**
  FPGA-based reconfigurable system
  + Data-flow computing, and its new architecture (CGRA)
  + Neuromorphic computing (like brain, e.g. Spiking NN)
  + Near-X computing (near-sensor, near-network, near-memory)
- **Members**
  - ✓ PI: Kentaro Sano
  - ✓ Researchers(4): Tomohiro Ueno, Jens Huthmann, Takaaki Miyajima, Atsushi Koshiba
  - ✓ Trainees: 4 students from Universities
  - ✓ Assistants: Tomoko Nakashima, Keiko Inaba

We are hiring!

Background

- **Mainstream architecture: many-core**
  (perf) = (# cores) x (freq) x (utilization)
- **Can many-core scale forever?**
  - ✓ **# of cores** limited by end of Moore's law
  - ✓ **frequency** limited by end of Dennard scaling
  - ✓ **utilization** limited by Neumann architecture and inefficient data movement



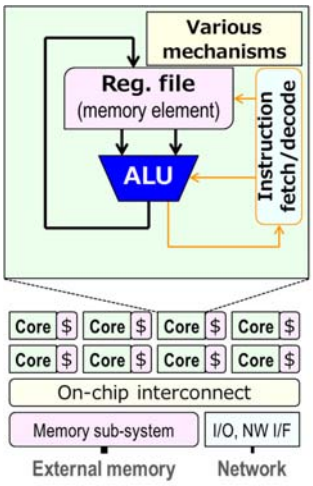
Performance scaling will end soon for general-purpose many-core processors.

- **New approaches are required.**
  - ✓ Domain-specific, specialized (optimized)
  - ✓ Computing in space, instead of computing in time
  - ✓ More localized, less centralized

Customization

Spatial computing with data-flow

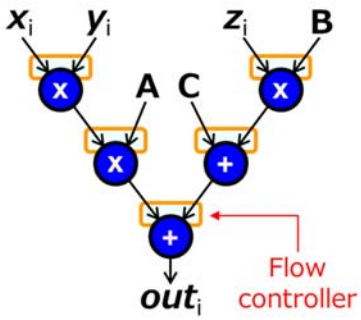
Spatial Custom Computing with FPGAs



Many-core architecture

- Sequentially executing with common operators (temporal computing)
- Then, parallelize it (pipelines, cores, ...)

- Spatially expanding operations into a circuit (as a data-flow graph)
- Flow data through data-flow circuits, to execute ops in parallel



FPGA

✓ Reconfigurable computing to customize circuits

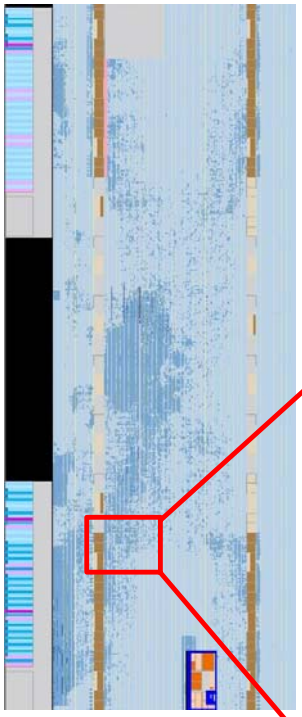
Data-flow architecture, customizable with FPGAs

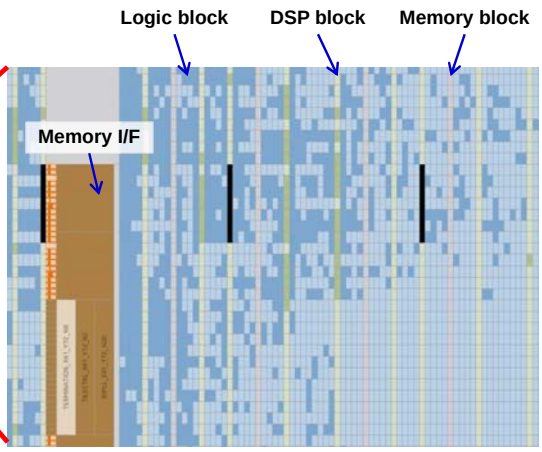
Non-FPGA data-flow architecture >> CGRA

What's FPGA?

- **Semiconductor device to construct arbitrary circuits**
  - ✓ Circuit reconfigurable device
  - ✓ Component for HW development
- **Array of hardware "blocks"**
  - ✓ Logic block
  - ✓ Memory block
  - ✓ DSP block (integer, floating-point)
  - ✓ I/O block (mem I/F, PCIe, HSSI)
  - ✓ Wires, clock trees, etc.
- **Recent FPGA : system-on-chip**

We can prototype own architecture.

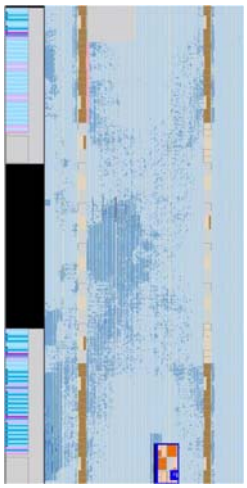




Industrial Trend of FPGAs

- **Use cases in data-center** — **Microsoft catapult**
- **Use cases in cloud-computing** — **AWS EC2 F1 instance**
- **Application development** — **Many companies** (genome sequence, security, ML…)

Why HPC with FPGAs?



- **Recent FPGA** (Intel Stratix10 in 14nm) **has lower-power, and**
  - ✓ Fast computing with 5760 FP-DSPs — 7TF at 450MHz
  - ✓ Fast external memories (DDR4, HBM2) — equivalent to CPUs, GPUs
  - ✓ Fast off-chip communication — 100Gbps x N serial links
  - ✓ Fast on-chip memories — ~1000 TB/s
- **Next-generation FPGA** (Intel Agilex in 10nm)

Customized circuits allow us to exploit them.

Research Activities

- **New architectures and approaches for future computing**

- **Data-flow architectures**
  - ✓ Reconfigurable computing with FPGAs
    - ➢ **FPGA cluster**
    - ➢ Hardware algorithm customized for each computing problem
  - ✓ Coarse-grained reconfigurable array (CGRA) processor

- **New computing principles**
  - ✓ Machine learning, brain simulation, Spiking neural network

R-CCS budget

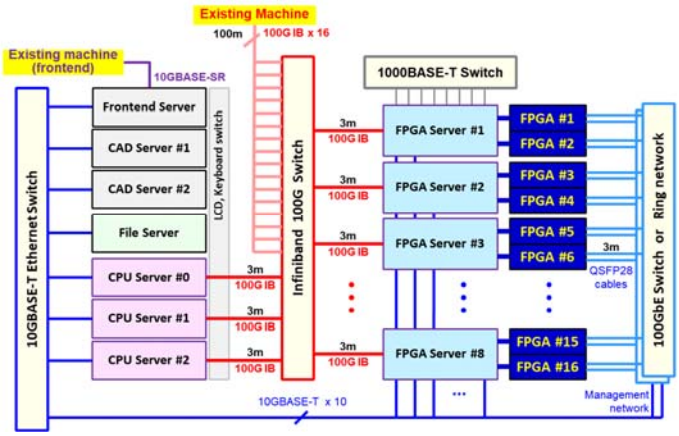
KAKENHI

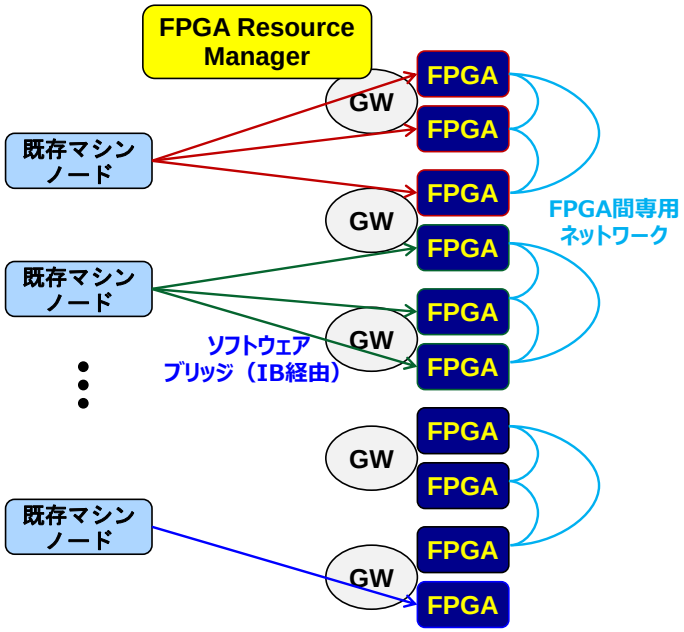
NEDO

Other external / internal budgets

FPGA Cluster Development

FPGAクラスタの開発状況（1）





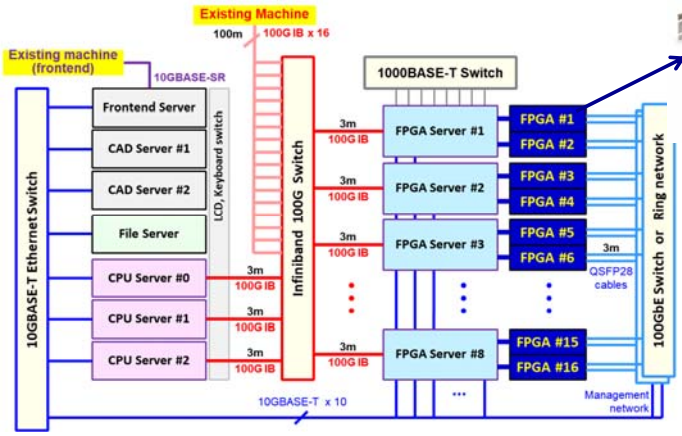
既存マシンノード（MPIプロセス）毎に自由に FPGAグループを割当て、タスクをオフロード

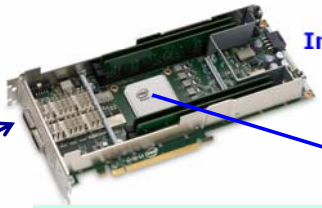
FPGAクラスタ試作機：

+ IB越しに遠隔FPGAを自由に利用可（資源割当のFlexibility）
+ FPGA間ネットワーク

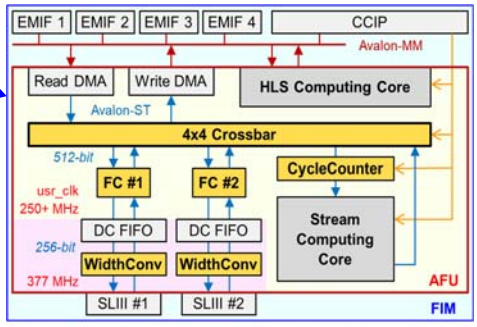
100Gbps 双方向リング　　実装完了・稼働中
100GbE スイッチネットワーク　　基盤実装完了・SoC組込

FPGAクラスタの開発状況（2）





Intel Stratix10 FPGA



リコンフィギャラブル システム・オン・チップ ver1：

+ カスタム部を自由に再構成可（3秒程度）
+ 基本機能（DMA, DDR4, クロスバ）、および計算コア

FPGAクラスタ試作機：

+ IB越しに遠隔FPGAを自由に利用可（**柔軟な資源割当**）
+ **FPGA間ネットワーク**

100Gbps 双方向リング　　実装完了・稼働中
100GbE スイッチネットワーク　　基盤実装完了・SoC組込

プログラミング & タスクオフロード：

+ ハードウェア（計算コア）

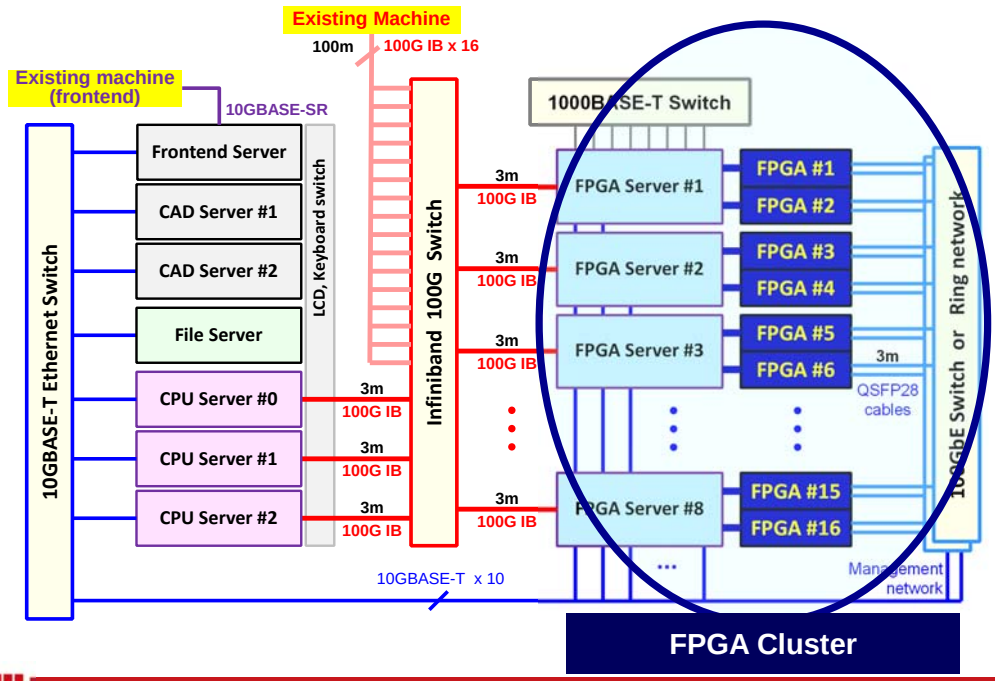
高位合成コンパイラ　（C, C++）
ドメイン特化コンパイラ（SPGen, etc.）
高抽象度ハードウェア記述（Chisel）

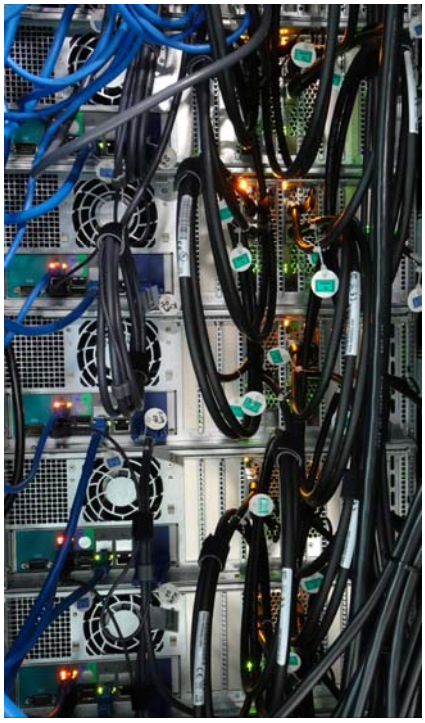
> まずは ハードウェアプログラマに 楽をさせるアプローチ（HWライブラリを開発）

+ ソフトウェア

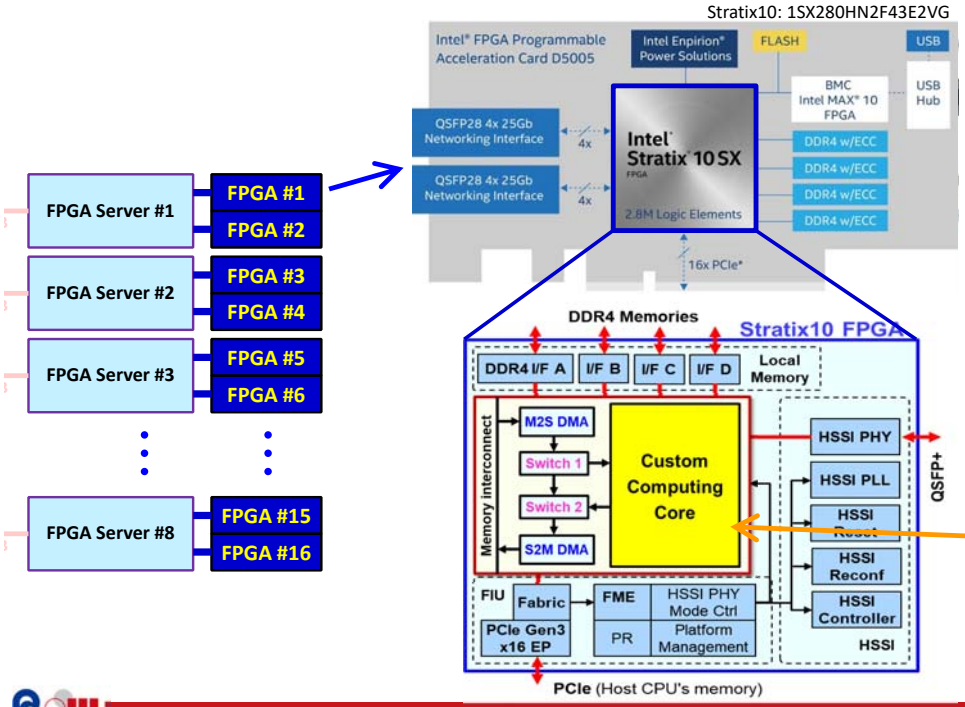
FPGAオブジェクト・クラス（デバイス制御, サービス）
Resource Manager（FPGA資源探索, ネットワーク設定, スケジュール）
コンパイラフロントエンド（OpenMP?, Task parallel?）

Extension System : FPGA Cluster





PAC : Programmable Acceleration Engine



Intel Stratix10 FPGA (14nm)

- ✓ *2753K LEs, 229 Mb BRAMs*
- ✓ *5760 FP DSPs (7TF @ 600MHz)*
- ✓ 8GB DDR4 x 4ch
- ✓ PCIe Gen3 x16
- ✓ 2x QSFP28 (100Gb/s)

FIM (FPGA Interface Manager)

- ✓ *Fixed hardware region*

AFU (Acceleration Function Unit)

- ✓ *Reconfigurable region*

AFU Shell : System for comp

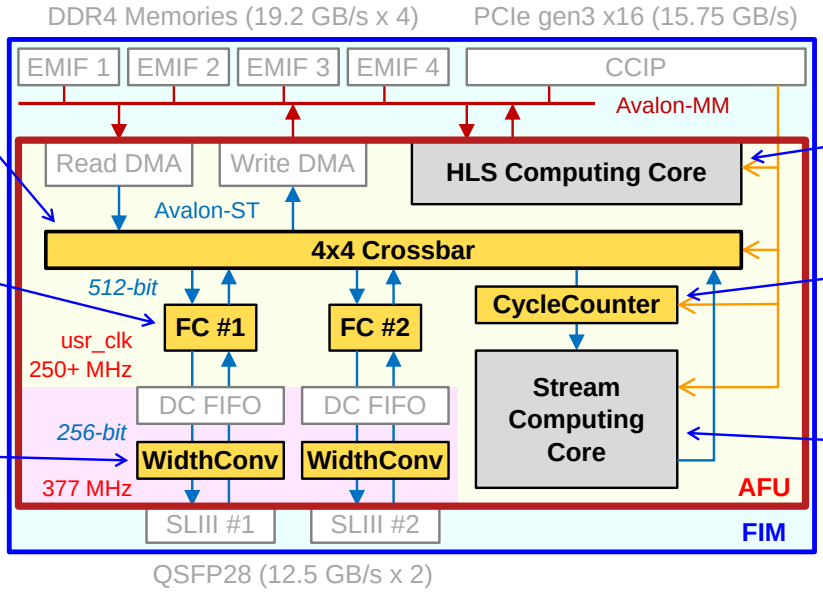
- ✓ *DMAs, Interconnect, Computing cores, etc.*

AFU Shell ver 1.0 with Hi-Speed Serial IF

4x4 Crossbar
- 4 input streams
- 4 output streams
- Programmable

Credit-based Flow Controller
- Enable back-pressure for end-to-end comm
- Own protocol w/ Cflit

Width Converter
- Convert width of stream at any ratio
- Configurable



HLS Computing Core
- Computing module synthesized by HLS (high-level synthesis) compiler

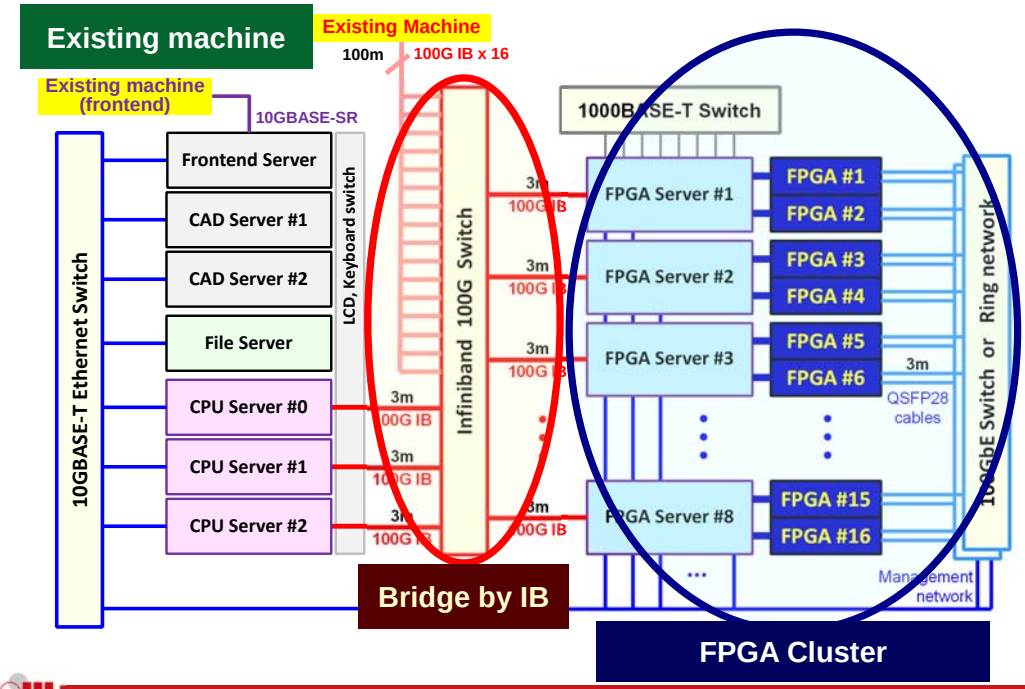
Cycle Counter
- Hardware performance counter

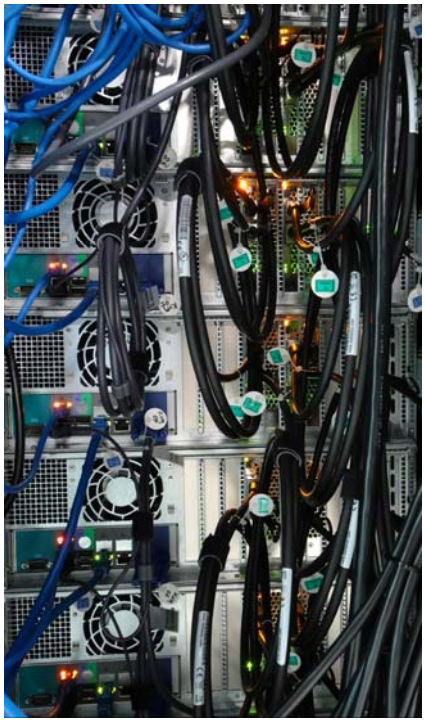
Stream Computing Core
- Computing module synthesized by SPGen (stream processor generator) compiler

System Operation

with FPGA Resource Allocation

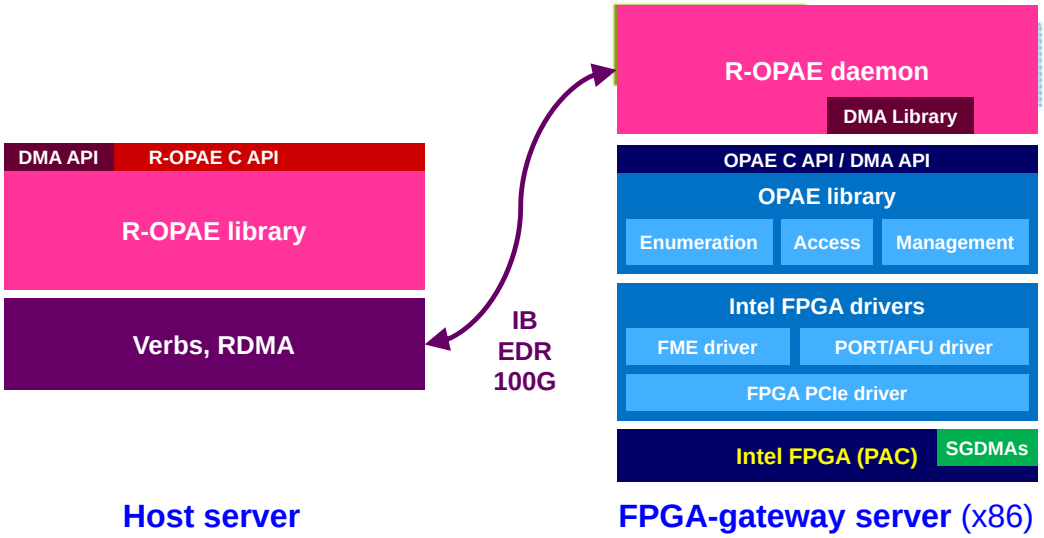
Extension System : FPGA Cluster





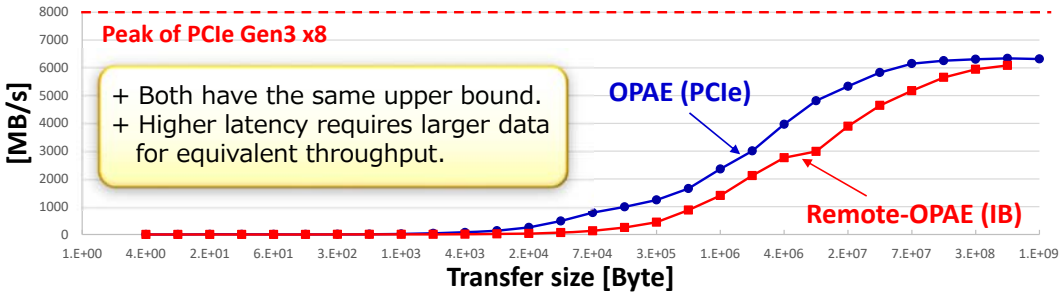
Remote-OPAE (Remote-FPGA APIs)

- **Software bridge for FPGA APIs & driver**
  - ✓ Intel Open Programmable Acceleration Engine (OPAE)
  - ✓ Infiniband EDR (100Gbps)
  - ✓ Can use any FPGAs in a system via IB network ***Very Flexible!***

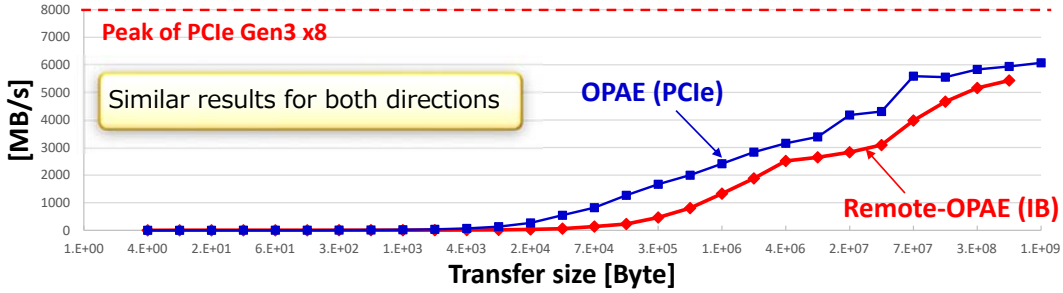


Throughput between Host and Arria10

Host to FPGA



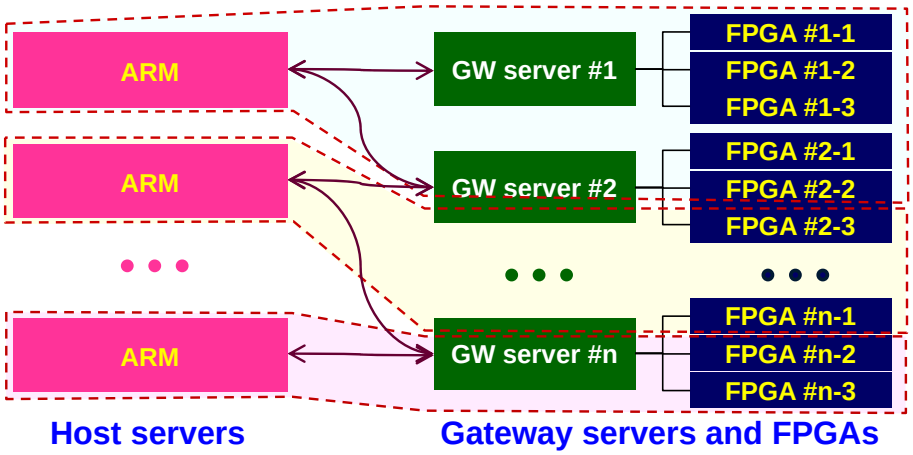
FPGA to Host



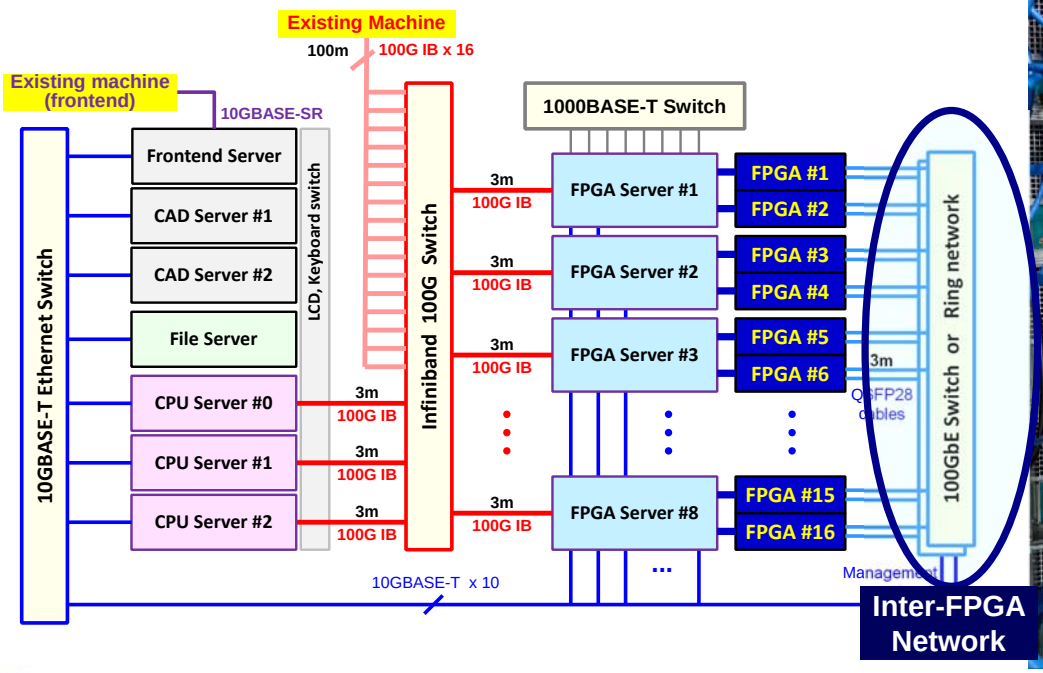
Advantages of Remote-OPAE

Software bridge provides high flexibility.

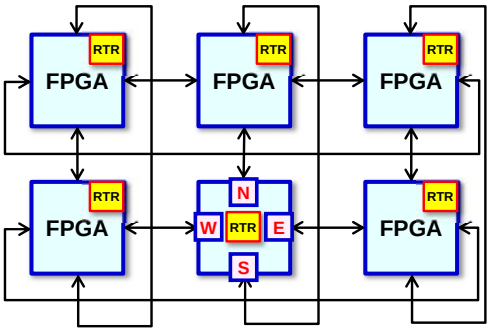
- ✓ Apps on hosts can use remote FPGAs transparently.
- ✓ Software-based **resource disaggregation**
- ✓ We can allocate hosts to arbitrarily groups of FPGAs.
- ✓ Vendor/ISA-independence



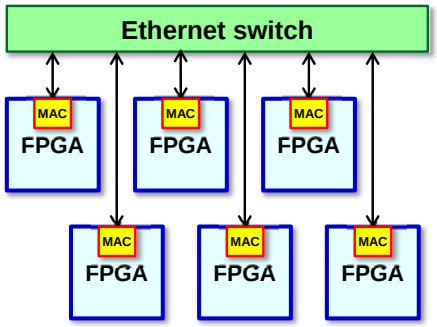
Dedicated Inter-FPGA Network



Two Approaches for Inter-FPGA Network



(Arbitrary topology virtualized)



Direct network : 2D torus

Pros) Smaller overhead (lower/fixed latency), easy to use

Cons) Inflexibility of resource allocation, more HW resources in SoC, difficulty to catch up

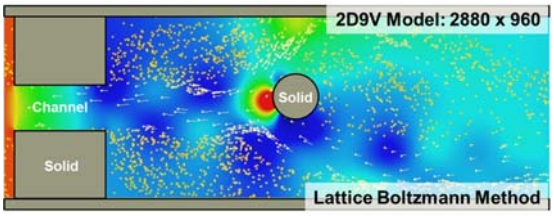
Indirect network : Ethernet

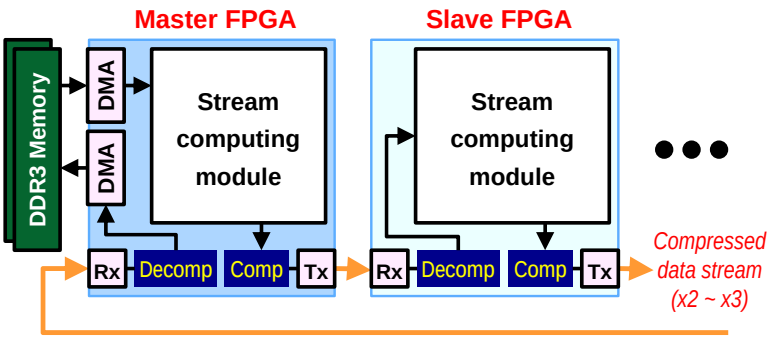
Pros) Flexibility of resource allocation, easy adoption of cutting-edge tech

Cons) Overhead of ethernet frames (higher and variable latency), difficulty in flow-control and use, cost of expensive switches

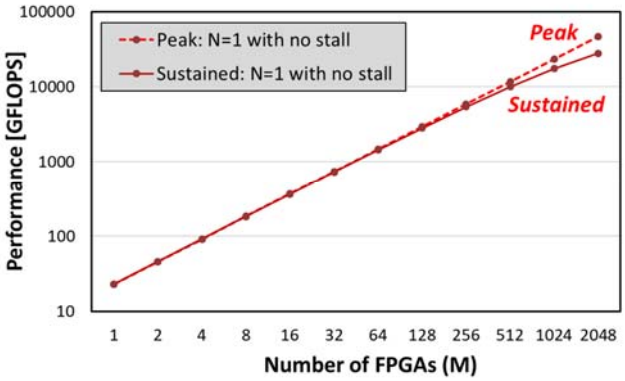
Ringed FPGAs for Deeper Pipelining

- **Deeply-pipelined FPGA cluster (1D ring)**
  - ✓ Linear array of Arria10 FPGAs
  - ✓ Lossless compressor for bandwidth extension
  - ✓ Pipelining works well for almost linear speedup if data stream is sufficiently large.





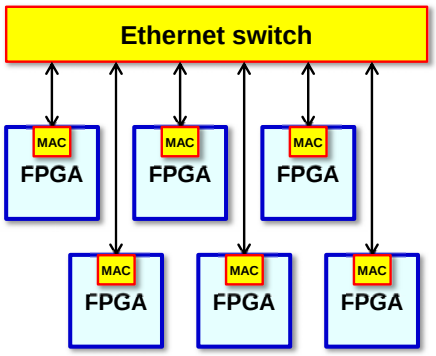
Block diagram of FPGAs in a ring

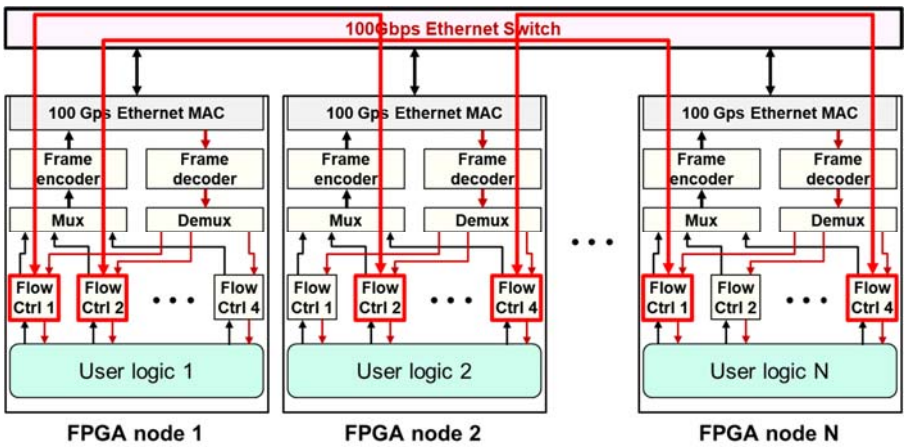


Performance estimation with a model

Ethernet-based NW for Flexibility (under development)

- **Inter-FPGA connection should also be flexible.**
  - ✓ ***Virtualized circuit switching network*** : form any topology with any groups of FPGAs.





100G Ethernet switches

Pros) Flexibility, cutting-edge technology

Cons) Overhead of ethernet frames, higher and variable latency, difficulty in flow-control and use

System Software

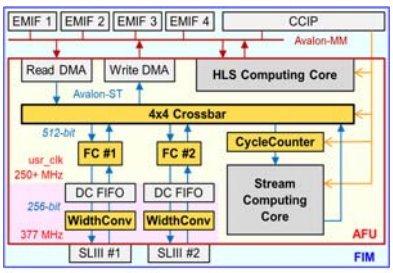
Hardware & Software Stack

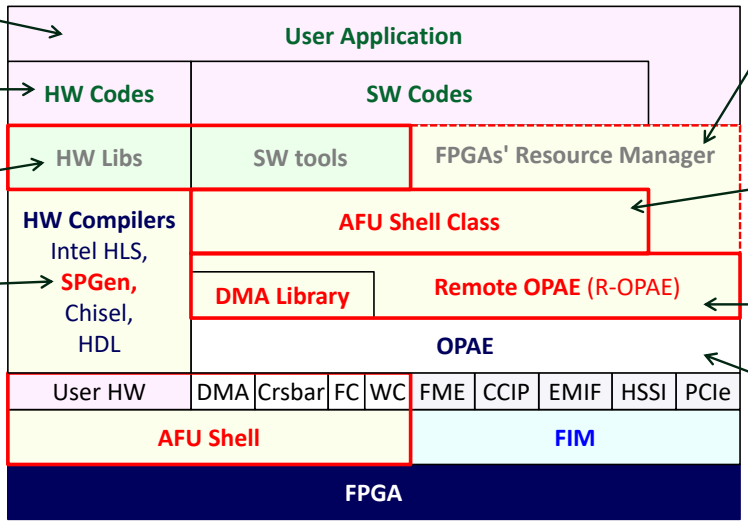
High-level application
- Including both SW & HW

Low-level application
- Separately-written HW & SW

Hardware, software libs/tools
- Pre-implemented functions

Programming tools
- SPGen : DSL for stream/ systolic computing





Resource manager
- Search and allocate resources of multiple FPGAs
- FPGA network management / control

AFU Shell class
- Object of AFU shell
- Abstraction of HW

Remote OPAE
- Software bridge using Infiniband Verbs

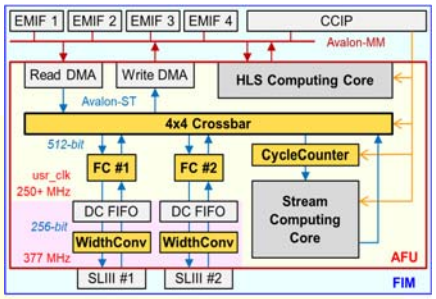
OPAE
- Low-level driver

AFU Shell Class Code Example

- **HW details are abstracted.**
  - ✓ Addresses of modules
  - ✓ Interface of service functions
  - ✓ Low-level R/W still possible.
- **App code is written simply.**
  - ✓ Instantiate AFUSH object
  - ✓ Open object
    - ➢ Search resources, then open
  - ✓ Use modules / services
    - ➢ Crossbar
    - ➢ Hardware cycle-counter
    - ➢ DMA (Host-FPGA, FPGA-FPGA)
    - ➢ Computing core
  - ✓ Close object

```
int very_simple_example(void)
{
  uint32_t allCycles, validCycles, csr;
  uint64_t bytes = 1024*1024*64;
  char *begin_ptr = (char *)malloc(sizeof(char)*bytes);

  afush_class afush("AFUSH0", "AFUSH0:");   <- Instantiate object

  if (!afush.open(0))   <- Open device
    {
      cout << "+ " << afush.name << " was not opened. Abort\n";
      return 0;
    }

  afush.set_crossbar(CROSSBAR_RdmaSl3a_Sl3b2CompWdma, cout);   <- Set Crossbar
  afush.read_crossbar(cout);

  // Blocking DMA transfers
  afush.dmaTransfer((uint64_t)begin_ptr, 0x800000000, bytes, HOST_TO_FPGA);

  afush.reset_ccounter(cout);   <- Use cycle-counter
  afush.dmaTransfer(0x00000000, 0x200000000, bytes, FPGA_TO_FPGA);   DMA Transfer
  afush.read_ccounter(allCycles, validCycles, csr, cout);

  afush.dmaTransfer(0x00000000, (uint64_t)begin_ptr, bytes, FPGA_TO_HOST);

  // Read and write a csr of your module
  cout << "== " << afush.mod[afush::ENTIRE_SPACE] << "\n"; // See memory map of "
  uint32_t val1 = 0x1234ABCD, val2;                        // "int" is NG.
  afush.mod[afush::ENTIRE_SPACE].writeMMIO32(0x00000340, val1); // crossbar write
  afush.mod[afush::ENTIRE_SPACE].readMMIO32 (0x00000340, val2); // crossbar read

  afush.close(cout);   <- Close device
  free(begin_ptr);

  return 1;
}
```


Programming FPGAs in a Cluster



- **Different options for various needs**
  - ✓ More SW-oriented : **HLS** Describe HW algorithms in C
  - ✓ More HW-oriented : **Chisel** Describe abstracted RTL structure in software language
  - ✓ Limited, but easier : **SPGen** Describe statements for stream-computing domain
- **OpenCL is also available.**
  - ✓ Ready for single-FPGA use
  - ✓ But, require extra effort to support inter-FPGA networks
  - ✓ We are discussing common I/F with Universities.

HLS: High-level synthesis, **Chisel**: Scala-based language for RTL, **SPGen** : Stream processor generator

Application examples

Simple Vector Add by Intel HLS Compiler

with a single Stratix10 FPGA (Intel PAC)

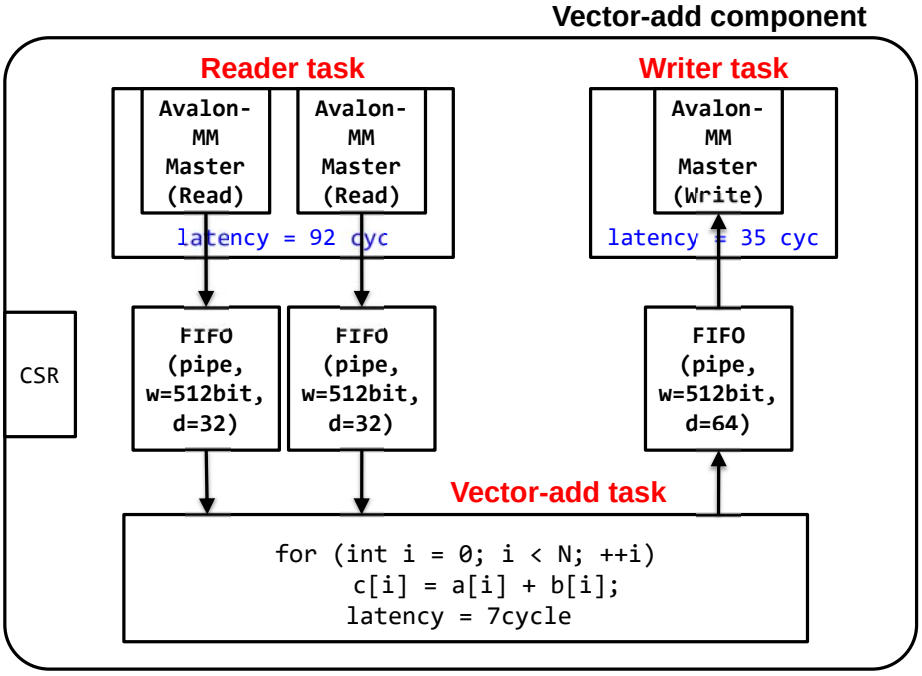
Programming for HLS Compiler

- **Write C Codes**
  - ✓ but for hardware structure and behavior
  - ✓ Need to know some level of hardware knowledge (types/configurations of port, latency, etc.)
- **Only for hardware side**
  - ✓ No software automatically generated for host CPU
  - ✓ Need to write host code manually, using afushell_class
- **However, we can do anything!**

Let's see example codes (made by Dr. Miyajima)

HLS Synthesized-Hardware Structure

- **Concurrently-working hardware modules**
  - ✓ Reader task module
  - ✓ Writer task module
  - ✓ Vector-add task module
- **Control-status registers (CSR)**
  - ✓ Control by hosts
- **90% of peak BW achievable**
  - ✓ Computing with regular memory accesses



HLS Code for Vector-add

```
struct pipe_width_float
{
  float data[16];
};

// ihc::pipe
ihc::pipe<input_pipe_id<1>, pipe_width_float, pipe_depth> pipe_a;
ihc::pipe<input_pipe_id<2>, pipe_width_float, pipe_depth> pipe_b;
ihc::pipe<input_pipe_id<3>, pipe_width_float, pipe_depth> pipe_c;

// Avalon-MMでDDRからデータを読んできて、FIFOに入れるtask
void mm_reader(avmm_master_port<float, 1> *a,
               avmm_master_port<float, 2> *b,
               uint32_t N, uint32_t stride_a, uint32_t stride_b)
{
#pragma unroll
  for (uint32_t i = 0; i < N; i += pipe_width)
    {
      pipe_width_float inputA, inputB;
#pragma unroll
      for (uint32_t j = 0; j < pipe_width; j++)
        {
          uint32_t index = (i + j);
          inputA.data[j] = (*a)[index * stride_a];
          inputB.data[j] = (*b)[index * stride_b];
        }
      pipe_a.write(inputA); // 16個データが来るまで待つことになる
      pipe_b.write(inputB);
    }
}
```

```
// FIFOからデータを取り出して、Avalon-MMでDDRに書き込むtask
void mm_writer(avmm_master_port<float, 3> *c,
               uint32_t N, uint32_t stride_c)
{
#pragma unroll
  for (uint32_t i = 0; i < N; i += pipe_width)
    {
      pipe_width_float outputC = pipe_c.read();
      for (uint32_t j = 0; j < pipe_width; j++)
        {
          uint32_t index = (i + j);
          (*c)[index * stride_c] = outputC.data[j];
          printf("writer: %d, %d\n", j, index * stride_c);
        }
    }
}

// 2つのFIFOからデータを読み出し、足し算して別のFIFOに書き出すtask
void vector_add_part5_stride(uint32_t N)
{
#pragma unroll
  for (uint32_t i = 0; i < N; i += pipe_width)
    {
      pipe_width_float inputA = pipe_a.read();
      pipe_width_float inputB = pipe_b.read();
      pipe_width_float outputC = {0};
#pragma unroll
      for (uint32_t j = 0; j < pipe_width; j++)
        {
          outputC.data[j] = inputA.data[j] + inputB.data[j];
        }
      pipe_c.write(outputC);
    }
}
```


HLS Code for Vector-add 2 of 2

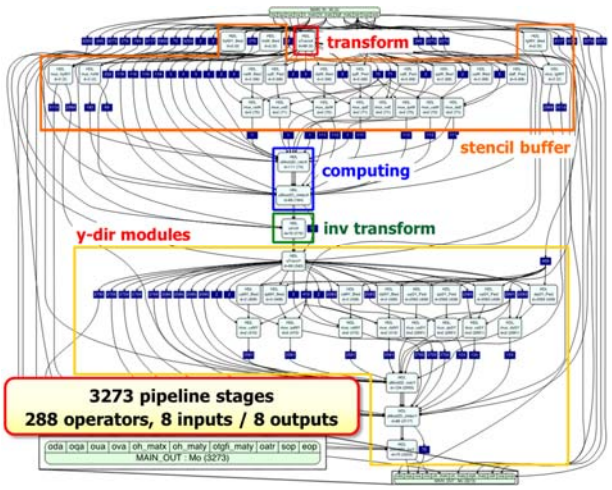
```
component hls_avalon_slave_component
void mm_master_component(
  hls_avalon_slave_register_argument avmm_master_port<float, 1> &a,
  hls_avalon_slave_register_argument avmm_master_port<float, 2> &b,
  hls_avalon_slave_register_argument avmm_master_port<float, 3> &c,
  hls_avalon_slave_register_argument uint32_t N,
  hls_avalon_slave_register_argument uint32_t stride_a,
  hls_avalon_slave_register_argument uint32_t stride_b,
  hls_avalon_slave_register_argument uint32_t stride_c)
{
  ihc::launch<mm_reader              >(&a, &b, N, stride_a, stride_b);
  ihc::launch<mm_writer              >(&c, N, stride_c);
  ihc::launch<vector_add_part5_stride>(N);

  ihc::collect<vector_add_part5_stride>();
  ihc::collect<mm_writer              >();
  ihc::collect<mm_reader              >();
}
```

```
int main(int argc, char **argv)
{

  unsigned int TEST_SIZE = std::atoi(argv[1]);
  const uint32_t stride_size = 2;
  const uint32_t elem_size = TEST_SIZE * stride_size;

  float A[elem_size];
  float B[elem_size];
  float C[elem_size];

  // mm_master interface class instances
  avmm_master_port<float, 1> mm_A(A, sizeof(float) * elem_size);
  avmm_master_port<float, 2> mm_B(B, sizeof(float) * elem_size);
  avmm_master_port<float, 3> mm_C(C, sizeof(float) * elem_size);

  // prepare the input data
  for (uint32_t i = 0; i < elem_size; i++)
    {
      A[i] = static_cast<float>(i); //rand();
      B[i] = static_cast<float>(i); //rand();
    }

  // Run the component
  mm_master_component(mm_A, mm_B, mm_C, TEST_SIZE,
                      stride_size, stride_size, stride_size);

  return 0;
}
```

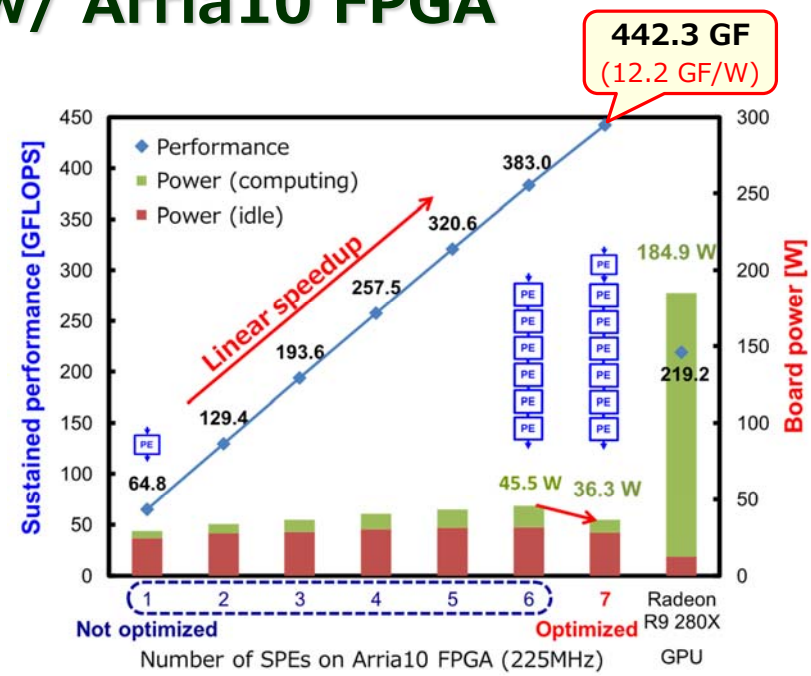
2D Tsunami Simulation by Data-Flow Compiler (SPGen)

with a single Arria10 FPGA (20nm), and
with multiple ringed Arria10 FPGAs

Tsunami Simulation w/ Arria10 FPGA



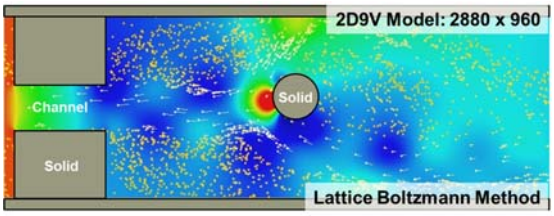
Stream Processing Element of Tsunami Kernel
(pipeline based on data-flow)

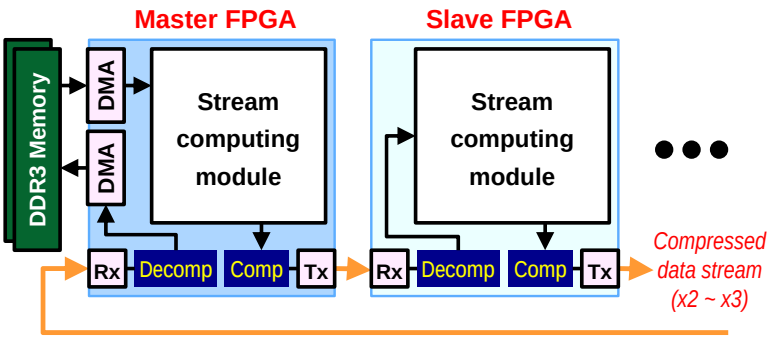


We expect 3+ TF per single Stratix10 FPGA.

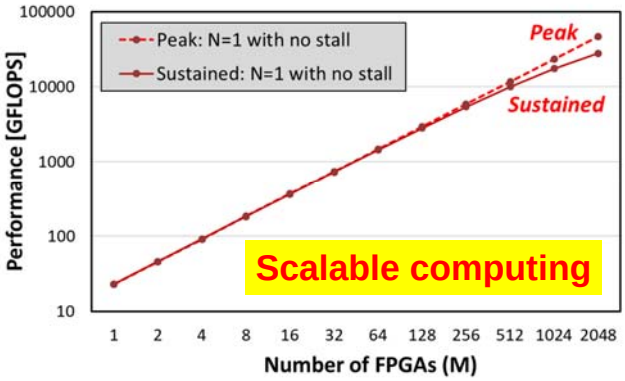
Ringed FPGAs for Deeper Pipelining

- **Deeply-pipelined FPGA cluster (1D ring)**
  - ✓ Linear array of Arria10 FPGAs
  - ✓ Lossless compressor for bandwidth extension
  - ✓ Pipelining works well for almost linear speedup if data stream is sufficiently large.





Block diagram of FPGAs in a ring



Performance estimation with a model

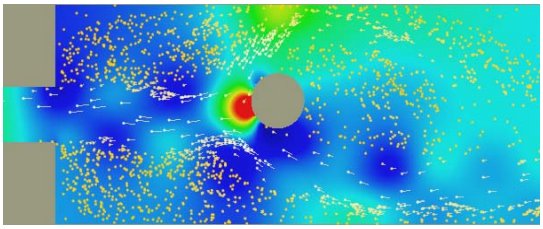
2D CFD based on Lattice Boltzmann Method by Data-Flow Compiler (SPGen)

(preliminary evaluation with a single Stratix10 FPGA)

Stream Computing with Stratix10 FPGA

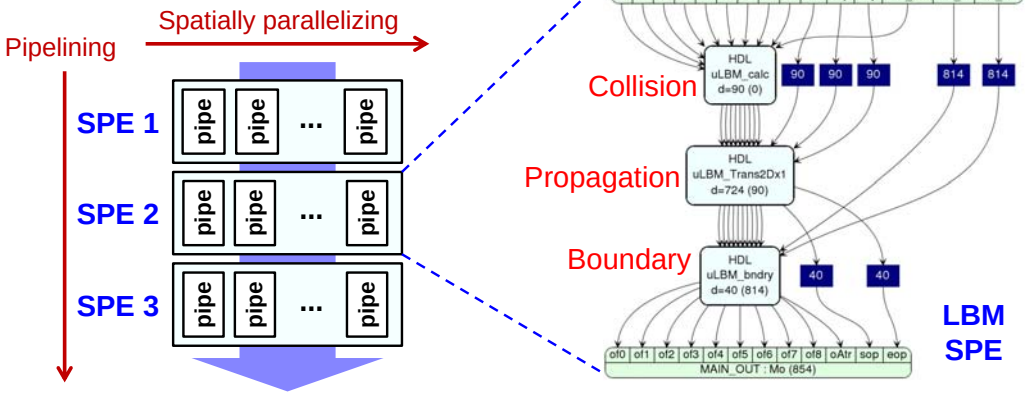
Fluid dynamics simulation

- ✓ **LBM** (lattice Boltzmann method)

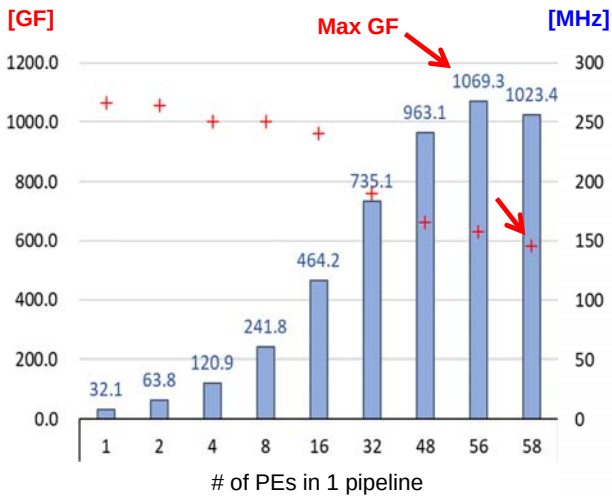


Array architecture of SPEs

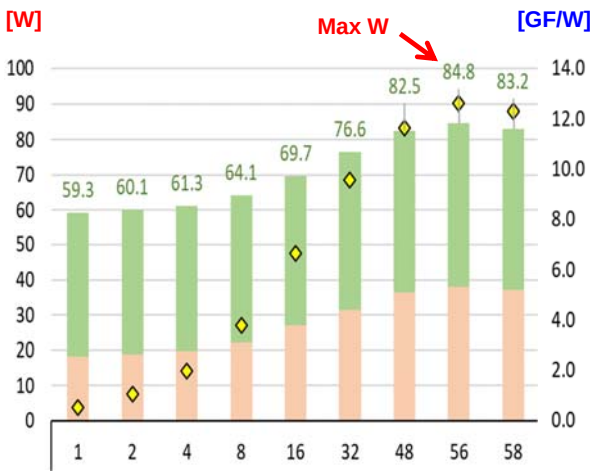
- ✓ **S**tream **P**rocessing **E**lements
- ✓ Generated by SPGen compiler



Performance and Power (LBM2D, S10)



Performance and Fmax



Power and Performance per Power

10% more PEs are expected. We target **450MHz for 3 TF+** performance.
We are also trying deeply-pipelined computing with a ring.

Target Benchmarks

◎ Very good ○ Good
△ Need investigation, but potential

	Problems	HW algorithm	HW operation	Efficient data-use	Low latency	Strong scaling
ハードウェアアルゴリズム予備設計済	Stencil computing	◎		○		○
	N-body (direct)	◎		○		○
ハードウェアアルゴリズム設計中・実装中	3D FFT	◎		◎		○
	Graph processing / Bayesian network	△	○	△		△
	Genome Seq matching	◎	◎	◎		○
	ML inference / training	○	◎	○		○
	Neuromorphic / Spiking NN	○	◎	○	○	○
	Data-base related	○	◎	○		○
	k-mean / SOM	◎	○	◎		◎

Future Work

- **Hardware**
  - ✓ On-chip router, RDMA module, to write DDR4 memories of other FPGA via NW
  - ✓ Ethernet-based network for FPGAs (base system was already developed.)
  - ✓ Increase Max frequency (250 ~ 350 MHz is okay, but 400 MHz is challenging.)
- **Programming**
  - ✓ Establish programming ~ exec flow for HLS and Chisel.
  - ✓ Higher-level abstraction / automation with C/C++ frontend
- **Benchmarking**
  - ✓ Implement target problems
  - ✓ Scale performance for multiple FPGAs
  - ✓ Evaluate performance and power

Opportunities for Collaboration

- **Seeds**
  - ✓ As research testbed,
    FPGA-cluster system ready for operation, especially with inter-FPGA network.
  - ✓ Fundamental software stacks (Driver, API, object class) are ready.
  - ✓ Can develop hardware in different abstraction levels.
    (e.g, we can try to accelerate your kernels by developing custom HW.)

- **Needs**
  - ✓ Need to develop system software (scheduler, manager, compiler frontend, etc.)
  - ✓ Need to extend/optimize/verify AFU shells.
  - ✓ Need applications to develop and evaluate with FPGAs.
    (Kernels like FFT, Big-data, AI, mixed-precision)

Need AT & AD?

Summary

- **Goal** Explore new processor architectures
  for next-generation HPC systems
  with their computing/programming model
- **Targets** FPGA-based reconfigurable system
  Data-flow computing, and its new architecture (CGRA)
  Neuromorphic computing (like brain, e.g. Spiking NN), etc.
- **FPGA cluster development for additional values of existing machines**
  - ✓ FPGA cluster ver1.0 is ready to use!
- **We call for collaboration in / out of R-CCS.**
  - ✓ Research teams, RIKEN centers, Universities, industries

And we are hiring!