Data-Flow Hardware Optimization by Design Space Exploration

Kentaro Sano

Processor Research Team, AICS, Riken

GSIS, Tohoku University

12 Mar, 2018

International Conference on Field-Programmable Technology

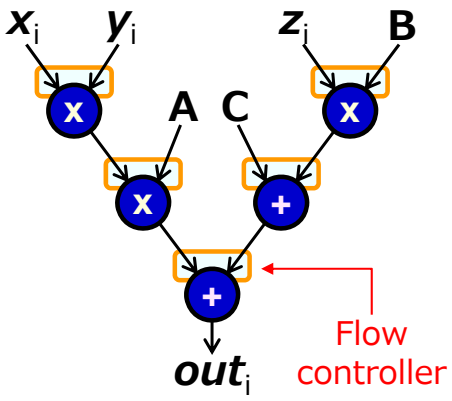
Naha, Okinawa, JAPAN, Dec 10-14, 2018

FPT'18 Naha

12 Mar, 2018

This Talk

- FPGA-based data-flow
- Data-flow compiler for FPGA
- Optimization for hardware mapping
- Case study
- Summary & future work



Data-flow computing

This is because …

Dark silicon and more expensive transistors

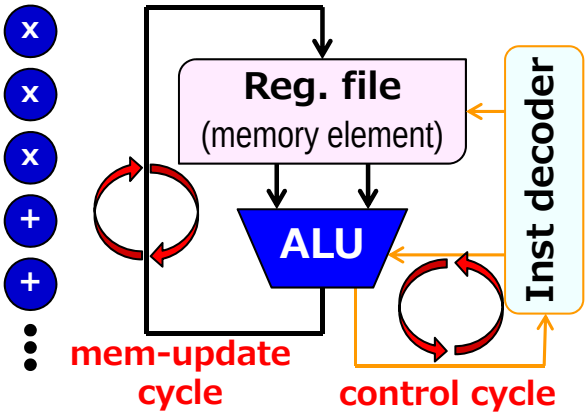
- ✓ 75% must be off for 7nm chip.

Latency-sensitive architecture

- ✓ von Neumann with
  + memory-update cycle
  + control cycle

Inefficient data movement between cores

- ✓ Read and write of memory hierarchy (scratch pad / cache)



What we need ?

Dark silicon and more expensive transistors

- ✓ 75% must be off for 7nm chip.

More efficient use of transistor-switching for necessary computation

Latency-sensitive architecture

- ✓ von Neumann with
  + memory-update cycle
  + control cycle

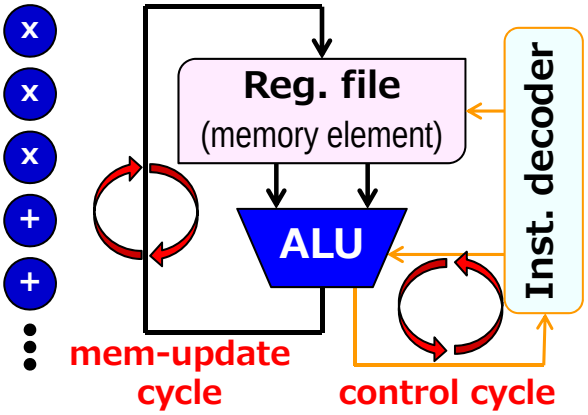
Latency-tolerant architecture w/o cycles

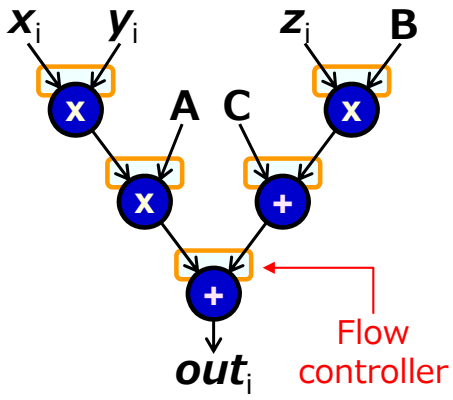
Inefficient data-movement between cores

- ✓ Read and write of memory hierarchy (scratch pad / cache)

Data-movement w/o memory access

Von-Neumann v.s. Data-Flow





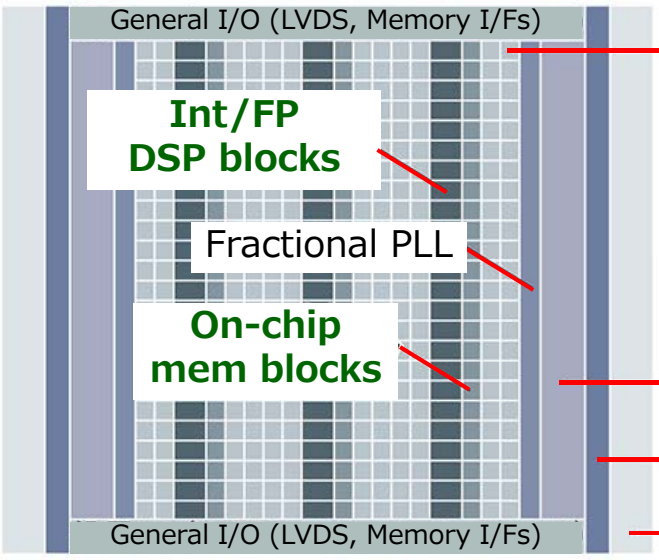
Many-core (Neumann)

- ✓ Cycles memory-update & on-the-fly control
- ✓ Memory hierarchy used for data movement & sync. between cores

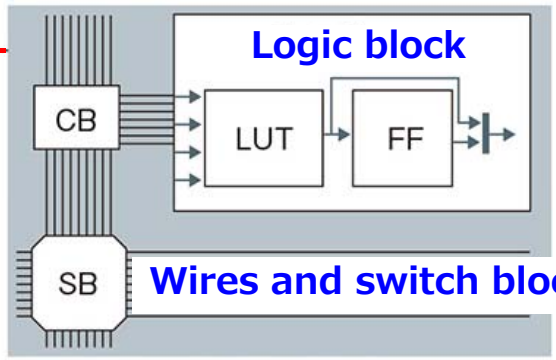
Data-flow

- ✓ Flow instead of cycles
- ✓ Structured control (determined before run)
- ✓ Direct connection for pipelining (latency tolerant)

FPGA : Platform for Custom Hardware



FPGA



High-speed I/O
(PCI-Express, 10G Ethernet
Interlaken, serial transceiver)
Memory I/F (DDR4, HBM, etc.)

Big array of floating-point DSPs, memories, and high-speed I/Os, rather than big array of logics

Goals of Research Project

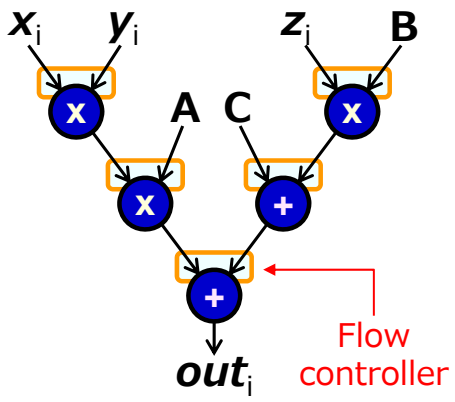
Data-flow HPC with FPGAs
promising in the upcoming Post-Moore Era

Post-Moore Era

- ✓ Many-core is not scaling, while a huge # of chips available

Promising solution

- ✓ **Data-flow** : latency tolerance (dependence is localized.)
- ✓ **FPGA** : customization & generalization

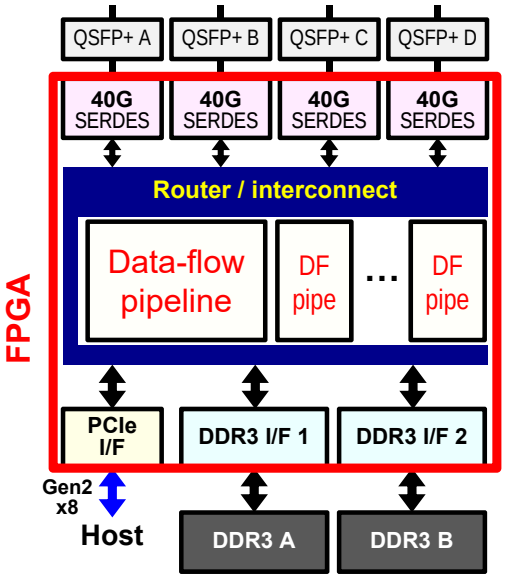


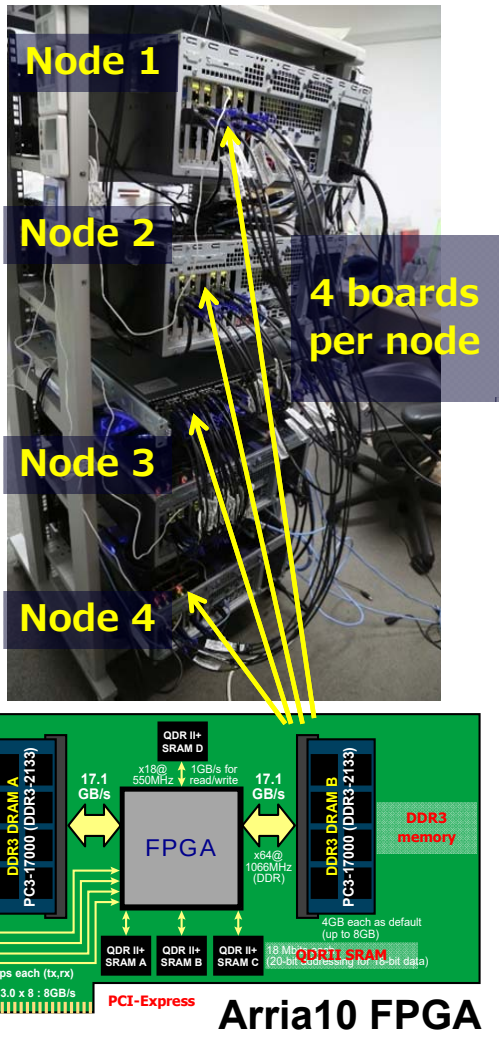
How to program and optimize data-flow for FPGA?

Prototype machine

Tightly-coupled FPGA cluster

- ✓ FPGA Shell
- ✓ System software stack
- ✓ Data-flow compiler for FPGA



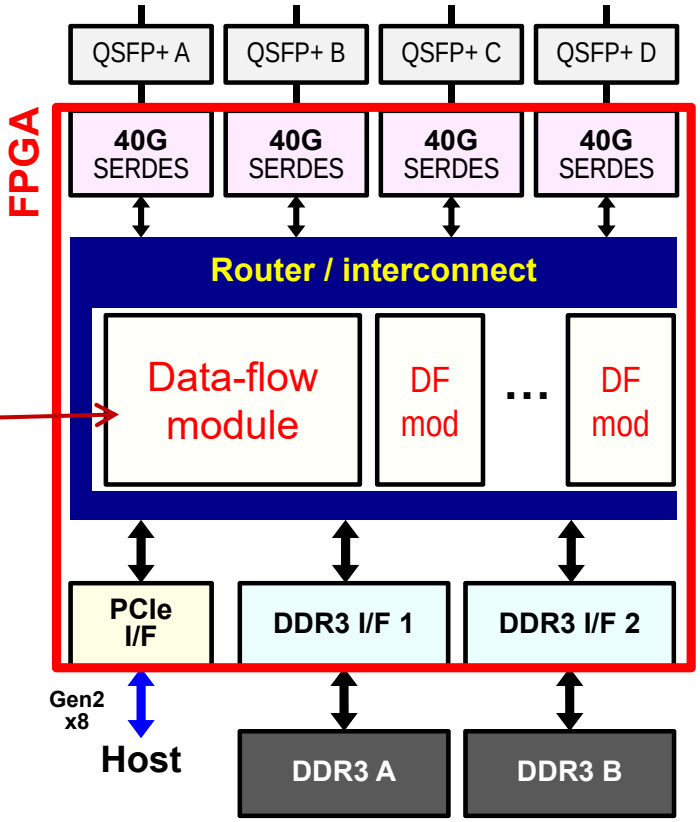


FPGA Shell

Fixed SoC HW on FPGA

- ✓ **Common functions**
  - ➢ Memory I/F, PCIe I/F, NW I/F, On-chip interconnect
- ✓ **Variable part : user HW**
  - ➢ Data-flow module
- ✓ **System software on host**
  - ➢ Driver, API library

FPGA Shell ver 2.2 is developed now.



Block diagram of FPGA Shell

Data-Flow Module

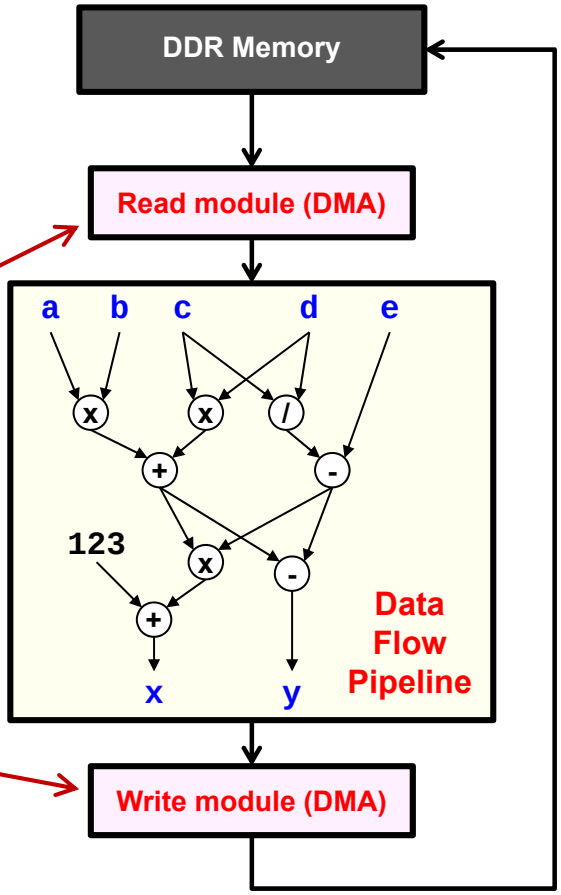
```
for (int j=0; j++; j<HIGHT)
 {
  for (int i=0; i++; i<WIDTH)
   {
    a = A[j][i];
    b = B[j][i];
    c = C[j][i];
    d = D[j][i];
    e = E[j][i];

    t1  = a*b + c*d;
    t2  = c/d - e;
    x   = t1*t2 + 123;
    y   = t1 - t2;

    X[j][i] = x;
    Y[j][i] = y;
   }
 }
```

Read memory

Compute

Write memory

Loop kernel code



Data-Flow Compiler : SPGen

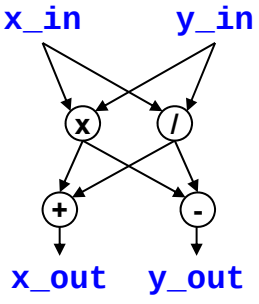
Stream Processor Generator

Formulae (Comp. DFG)

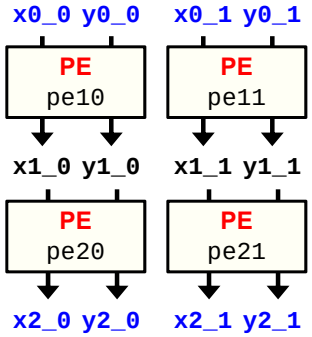
Module calls (HW Structure)

```
Name      PE;     ### Define pipeline "PE"
Main_In   {in:: x_in,  y_in};
Main_Out  {out::x_out, y_out};

EQU  eq1,  t1    = x_in * y_in;
EQU  eq2,  t2    = x_in / y_in;
EQU  eq3,  x_out = t1 + t2;
EQU  eq3,  y_out = t1 - t2;
```

SPD : Stream processing description




```
Name      Core;         ### Define IP core "Core"
Main_In   {in:: x0_0, x0_1, y0_0, y0_1};
Main_Out  {out::x2_0, x2_1, y2_0, y2_1};

### Description of parallel pipelines for t=0
HDL pe10, 123, (x1_0, y1_0) = PE(x0_0, y0_0);
HDL pe11, 123, (x1_1, y1_1) = PE(x0_1, y0_1);

### Description of parallel pipelines for t=1
HDL pe20, 123, (x2_0, y2_0) = PE(x1_0, y1_0);
HDL pe21, 123, (x2_1, y2_1) = PE(x1_1, y1_1);
```

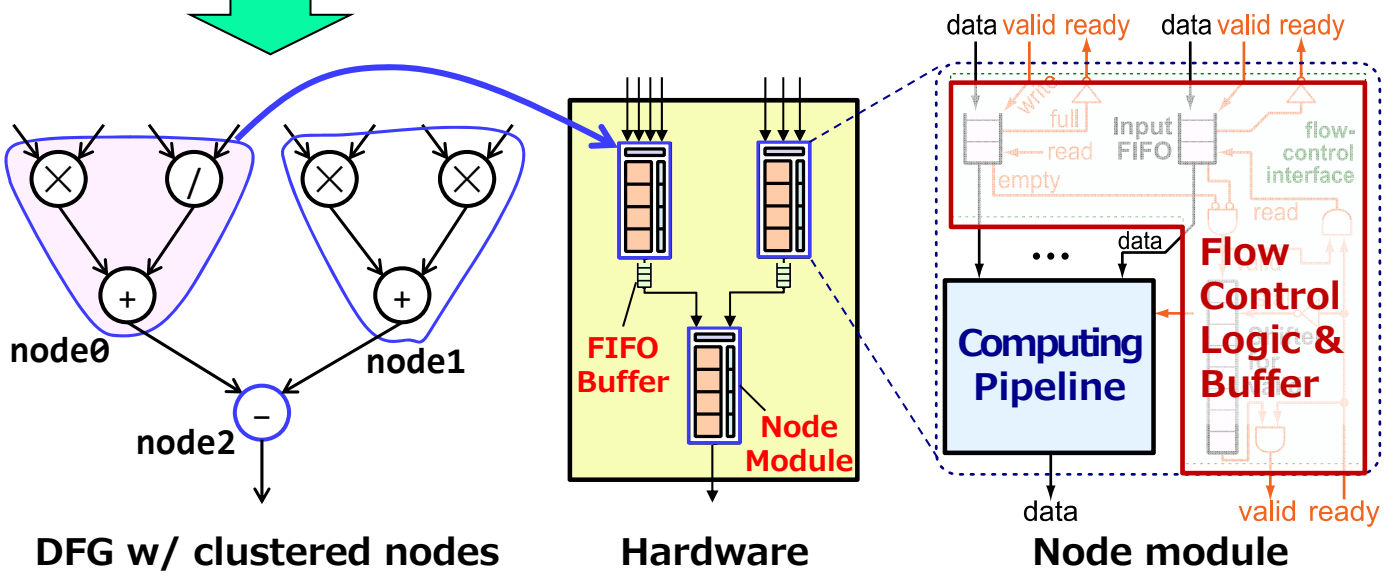


Hardware Generated by SPGen

SPD (Stream Processing Description)

```
EQU node0, out0 = in0*in1 + in2/in3;
EQU node1, out1 = in4*in5 + in6*in7;
EQU node2, out2 = out0 - out1;
```

Each line corresponds to **each node** in DFG.

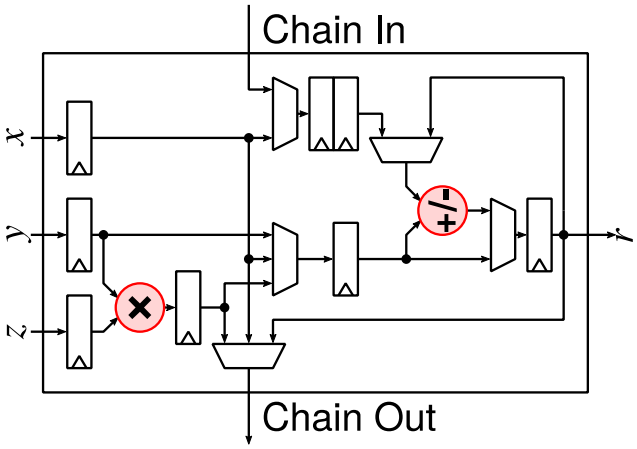


DFG w/ clustered nodes | Hardware | Node module

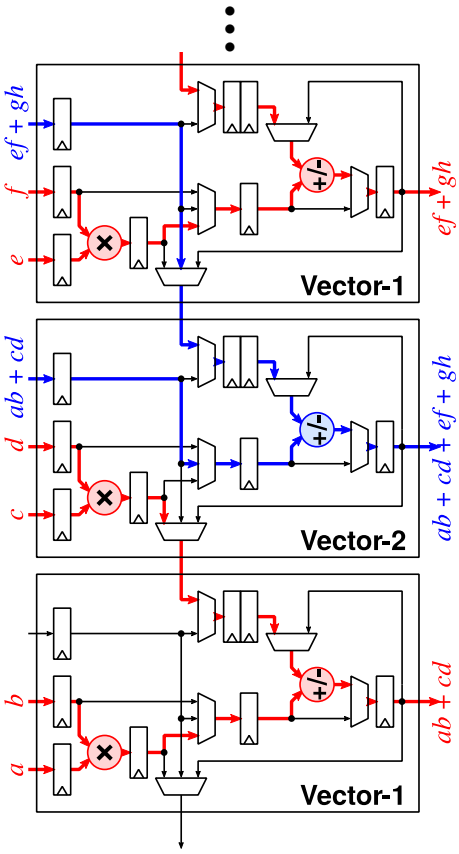
Problems of SPGen

Productivity issue for

- ✓ Floating-point DSP mapping
- ✓ Optimal node clustering

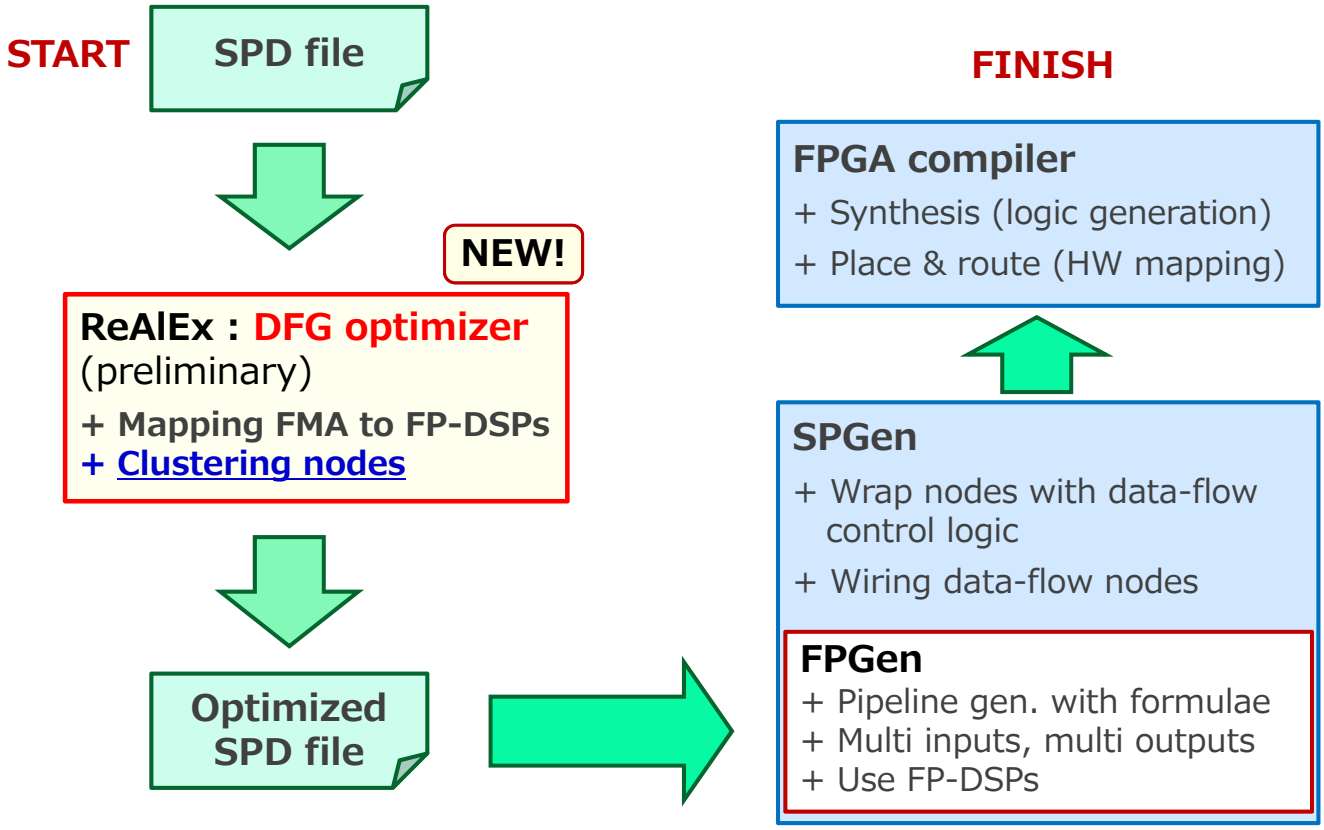


FP-DSP of Intel FPGA
(Add, Mul, or FMA)



FP-DSP chaining for vector operation

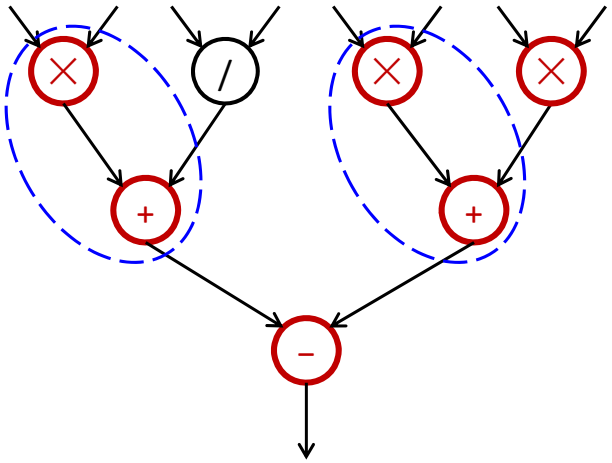
New Tool Chain with Optimization



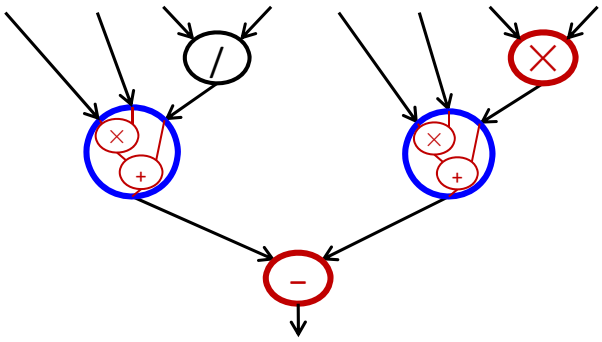
Floating-point DSP mapping

Saving DSPs with FMA & Chained Modes

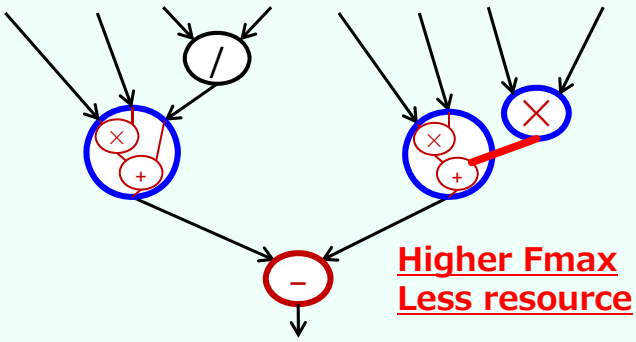
6 DSPs used



DFG only with single-op DSPs



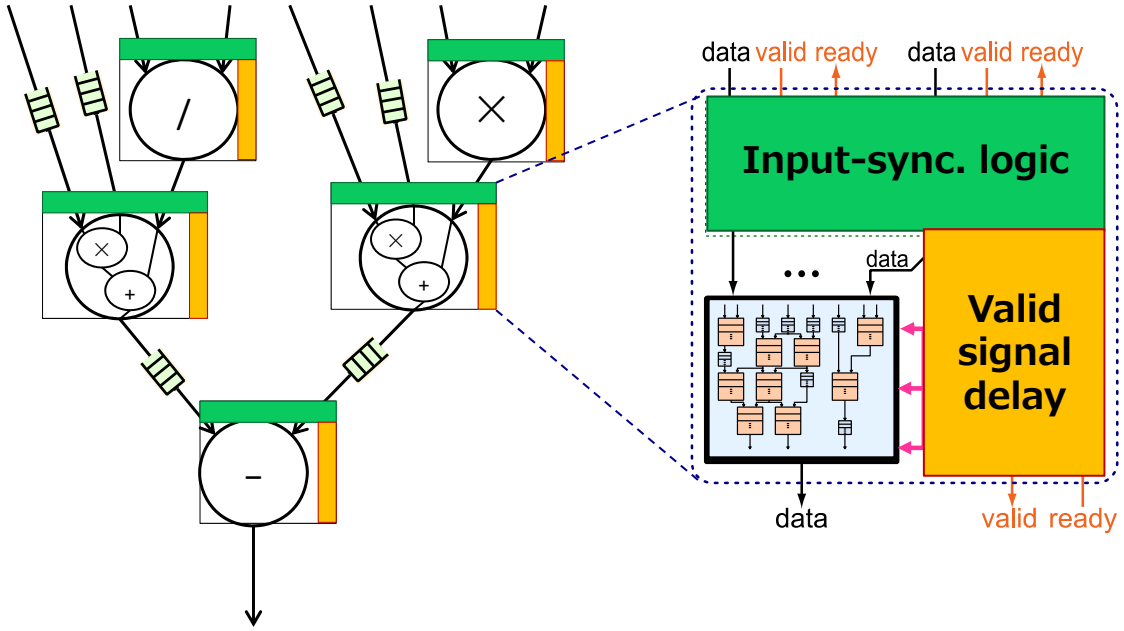
DFG w/ FMA DSPs (4 DSPs)



DFG w/ chained DSPs (4 DSPs)

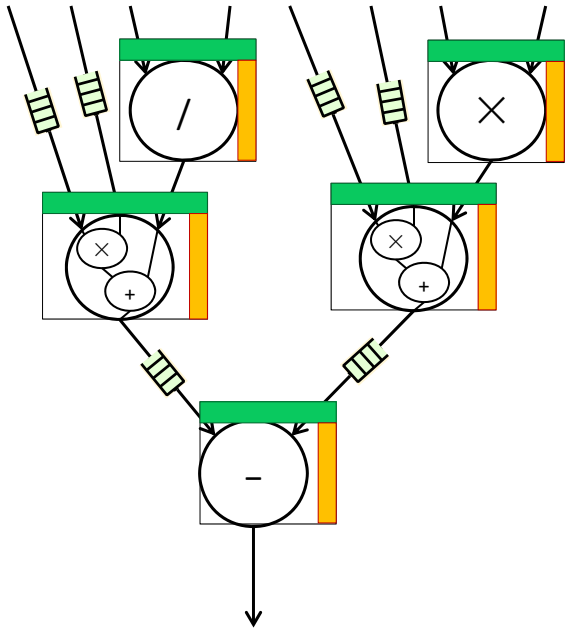
Optimal node clustering

Clustering Granularity v.s. Overhead

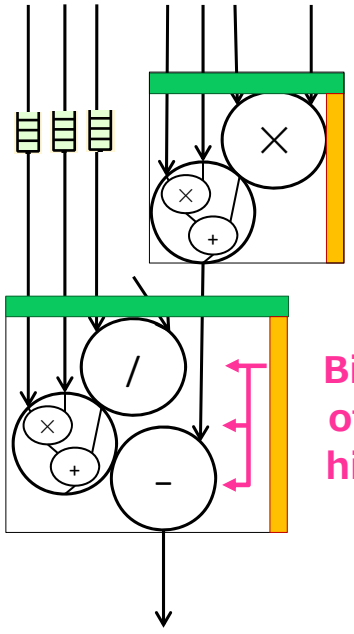


Fine-grain
(large overhead, high Fmax)

Tradeoff between Overhead & Fmax



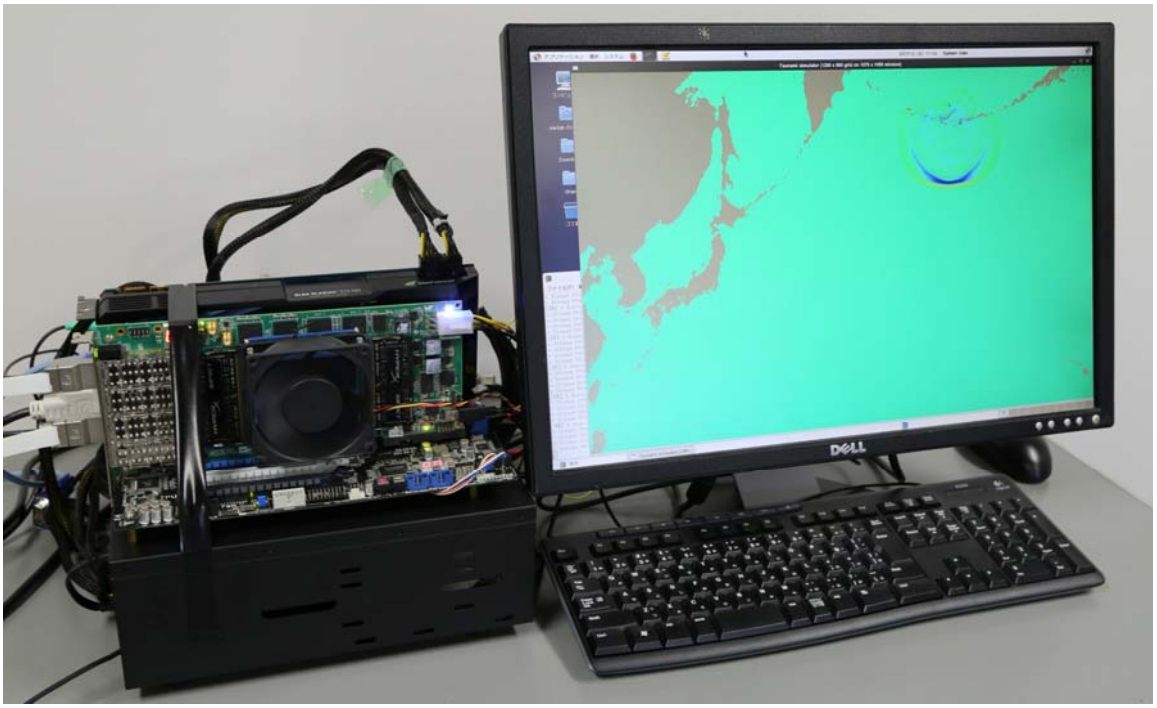
Fine-grain
(large overhead, high Fmax)



More inputs, higher sync. latency

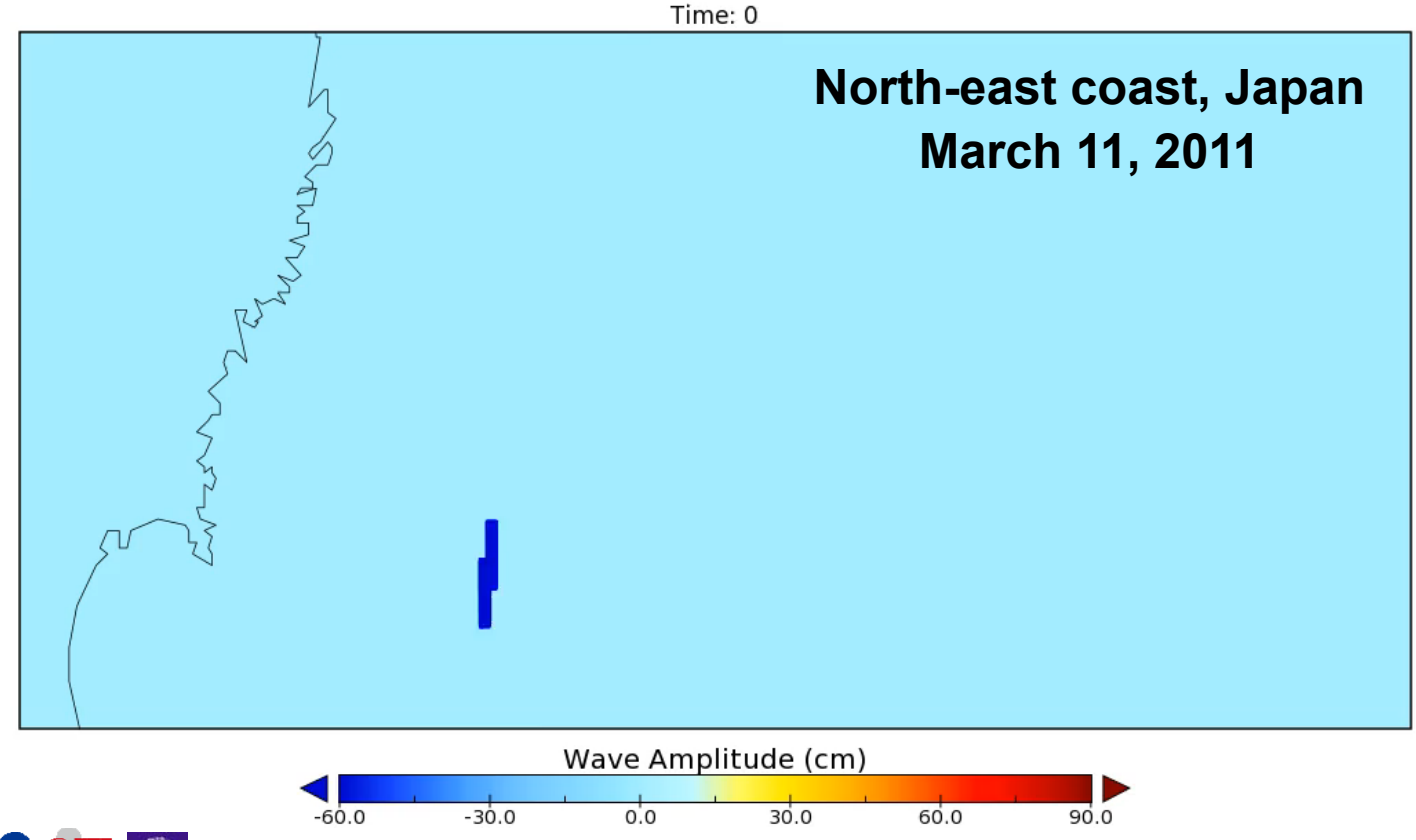
Bigger fan-out of stall signal, higher latency

Coarse grain
(small overhead, low Fmax)

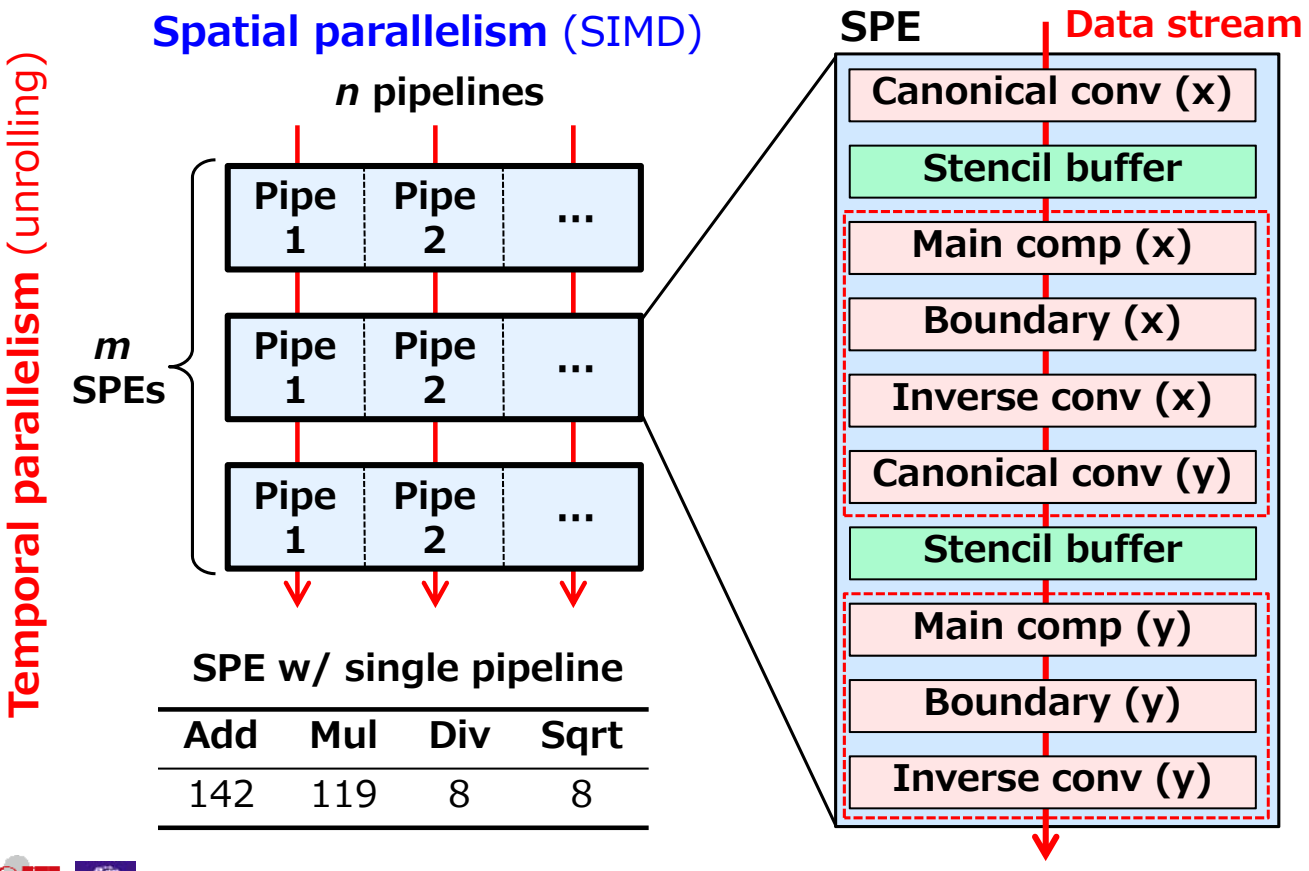


Tsunami Simulator with Arria10 GX1150 FPGA

Tsunami Simulation by using FPGA



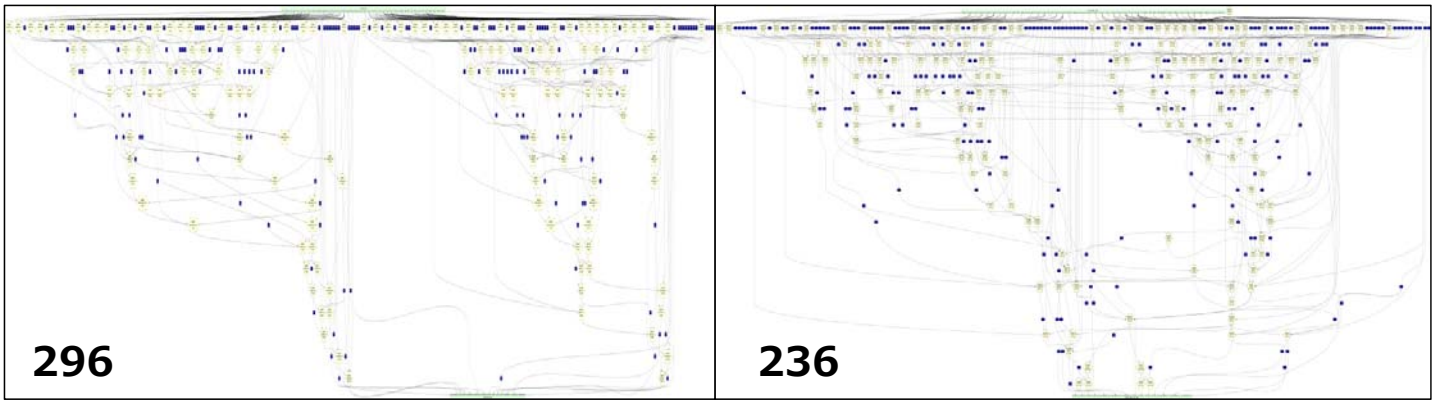
Stream Processing Element (SPE)

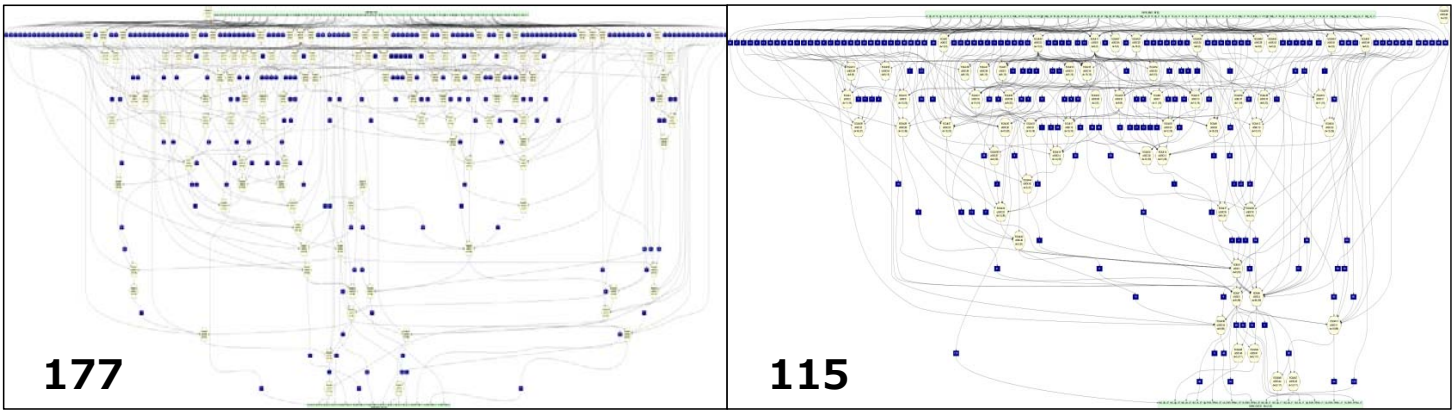


SPE w/ single pipeline

Add	Mul	Div	Sqrt
142	119	8	8

DFGs with different Num of Clusters





Summary

FPGA-based data-flow computing

- ✓ Efficient use of transistors
- ✓ Latency tolerance

Optimization in mapping DFG to HW resources

- ✓ Fused operations of DSP blocks
- ✓ Less overhead of flow-control logics
- ✓ Tsunami case study

Future work

- ✓ More sophisticated algorithm
- ✓ C to DFG frontend
- ✓ Stratix10 FPGA
- ✓ Automatic mapping to FPGAs connected in 2D torus