Dataflow Programming with MaxCompiler
Lecture Overview

- Programming DFEs
- MaxCompiler
- Streaming Kernels
- Compile and build
- Java meta-programming
Reconfigurable Computing with DFES

Xilinx Virtex-6 FPGA

Logic Cell ($10^5$ elements)

DSP Block

IO Block

Block RAM

Block RAM (20TB/s)
DFE Acceleration Hardware Solutions

MaxCard
PCI-Express Gen 2
Typical 50W-80W
24-48GB RAM

MaxNode
1U server
4 MAX3 Cards
Intel Xeon CPUs

MaxRack
10U, 20U or 40U Rack
How could we program it?

- Schematic entry of circuits
- Traditional Hardware Description Languages
  - VHDL, Verilog, SystemC.org
- Object-oriented languages
  - C/C++, Python, Java, and related languages
- Functional languages: e.g. Haskell
- High level interface: e.g. Mathematica, MatLab
- Schematic block diagram e.g. Simulink
- Domain specific languages (DSLs)
Accelerator Programming Models

Flexible Compiler System: MaxCompiler

Level of Abstraction

Possible applications

- DSL
- DSL
- Higher Level Libraries
- Higher Level Libraries
Acceleration Development Flow

1. **Start**
   - Original Application

2. Identify code for acceleration and analyze bottlenecks
3. Transform app, architect and model performance
4. Write MaxCompiler code
5. Integrate with Host code
6. Simulate

7. **NO**
   - Accelerated Application
   - Meets performance goals?
   - Build for Hardware
   - Functions correctly?

8. **YES**
   - Accelerated Application
MaxCompiler

- Complete development environment for Maxeler DFE accelerator platforms
- Write *MaxJ* code to describe the dataflow accelerator
  - *MaxJ* is an extension of Java for MaxCompiler
  - *Execute* the Java → *generate* the accelerator
- C software on CPUs *uses* the accelerator

```java
class MyAccelerator extends Kernel {
    public MyAccelerator(...) {
        DFEVar x = io.input("x", dfeFloat(8, 24));
        DFEVar y = io.input("y", dfeFloat(8, 24));

        DFEVar x2 = x * x;
        DFEVar y2 = y * y;
        DFEVar result = x2 + y2 + 30;

        io.output("z", result, dfeFloat(8, 24));
    }
}
```
Application Components

- CPU
- SLiC
- MaxelerOS
- Memory
- CPU
- SLiC
- MaxelerOS
- Memory

PCI Express

Host application

DFE

Kernels

Memory

Manager
Programming with MaxCompiler

- **User Input**: Host Application (*.c, *.f90 ...)
  - Compiler
  - Rewrite only code to be accelerated
  - MaxIDE
    - MyKernel.maxj
    - MyManager.maxj
  - MaxCompiler
    - MAX File Sim or H/W (.max)
- **Output**: Executable
- **Library**: SLiC Library
- **Library**: maxelerOS Library
for (int i = 0; i < DATA_SIZE; i++)
    y[i] = x[i] * x[i] + 30;

\[ y_i = x_i \times x_i + 30 \]
int *x, *y;
for (int i = 0; i < DATA_SIZE; i++)
    y[i] = x[i] * x[i] + 30;

Manager m = new Manager();
Kernel k =
    new MyKernel();
    m.setKernel(k);
    m.setIO(
        link("x", CPU),
        link("y", CPU));
    m.build();

DFEVar x = io.input("x", dfeInt(32));
DFEVar result = x * x + 30;
    io.output("y", result, dfeInt(32));
Development Process

**Host Code (.c)**

```c
int* x, *y;
MyKernel(DATA_SIZE, x, DATA_SIZE*4);
```

**MyManager (.maxj)**

```java
Manager m = new Manager();
Kernel k =
   new MyKernel();
m.setKernel(k);
m.setIO(
   link("x", CPU),
   link("y", DRAM_LINEAR1D));
m.build();
```

**MyKernel (.maxj)**

```java
DFEVar x = io.input("x", dfelnt(32));
DFEVar result = x * x + 30;
io.output("y", result, dfelnt(32));
```
public class MyKernel extends Kernel {

    public MyKernel (KernelParameters parameters) {
        super(parameters);

        DFEVar x = io.input("x", dfeInt(32));
        DFEVar result = x * x + 30;
        io.output("y", result, dfeInt(32));
    }
}

The Full Kernel
Streaming Data through the Kernel

5 4 3 2 1 0

x

0

x

0

+

30

y

30
Streaming Data through the Kernel

5 4 3 2 1 0

x

1

x

+ 30

31

y

30 31
Streaming Data through the Kernel

5 4 3 2 1 0

5

x

4

2

x

+ 30

34

y

30 31 34
Streaming Data through the Kernel

5 4 3 2 1 0

x

3

x

9

+

30

30 31 34 39
Streaming Data through the Kernel
Streaming Data through the Kernel
Compile, Build and Run

• Java program *generates* a MaxFile *when it runs*

1. Compile the Java into .class files
2. Execute the .class file
   - Builds the dataflow graph in memory
   - Generates the hardware .max file
3. Link the generated .max file with your host program
4. Run the host program
   - Host code automatically configures DFE(s) and interacts with them at run-time
Java meta-programming

• You can use the full power of Java to write a program that *generates* the dataflow graph

• Java variables can be used as constants in hardware
  – int y; DFEVar x; x = x + y;

• Hardware variables can not be read in Java!
  – *Cannot do*: int y; DFEVar x; y = x;

• Java conditionals and loops choose *how* to generate hardware ➔ not make run-time decisions
What dataflow graph is generated?

DFEVar x = io.input("x", type);
DFEVar y;

y = x + 1;

io.output("y", y, type);
What dataflow graph is generated?

DFEVar x = io.input("x", type);
DFEVar y;

y = x + x + x;

io.output("y", y, type);
Dataflow Graph Generation: Variables

What’s the value of $h$ if we stream in 1?

```java
DFEVar h = io.input("h", type);
int s = 2;

s = s + 5
h = h + 10

h = h + s;
```

What’s the value of $s$ if we stream in 1?

```java
DFEVar h = io.input("h", type);
int s = 2;

s = s + 5
h = h + 10

s = h + s;
```

Compile error.
You can’t assign a hardware value to a Java int.
Dataflow Graph Generation: Conditionals

**What dataflow graph is generated?**

DFEVar x = io.input("x", type);
int s = 10;
DFEVar y;

if (s < 100) { y = x + 1; }
else { y = x - 1; }

io.output("y", y, type);

**What dataflow graph is generated?**

DFEVar x = io.input("x", type);
DFEVar y;

if (x < 10) { y = x + 1; }
else { y = x - 1; }

io.output("y", y, type);

Compile error.
You can’t use the value of ‘x’ in a Java conditional
Conditional Choice in Kernels

- Compute both values and use a *multiplexer*.
  - $x = \text{control.mux}(\text{select, option0, option1, \ldots, optionN})$
  - $x = \text{select} \ ? \ \text{option1} : \text{option0}$

Ternary-if operator is overloaded

```
DFEVar x = io.input("x", type);
DFEVar y;
y = (x > 10) ? x + 1 : x - 1
io.output("y", y, type);
```
What dataflow graph is generated?

DFEVar x = io.input("x", type);
DFEVar y = x;
for (int i = 1; i <= 3; i++) {
    y = y + i;
}
io.output("y", y, type);

Can make the loop any size – until you run out of space on the chip!
Larger loops can be partially unrolled in space and used multiple times in time – see Lecture on loops and cyclic graphs
Real data flow graph as generated by MaxCompiler
4866 nodes;
10,000s of stages/cycles
Exercises

1. Write a MaxCompiler kernel program that takes three input streams $x$, $y$ and $z$ which are dfeInt(32) and computes an output stream $p$, where:

\[
p_i = \begin{cases} 
(x_i + y_i) + 2 & \text{if } z_i > x_i \\
(x_i + z_i) \times 2 & \text{if } z_i < x_i \\
(x_i \times y_i \times z_i) & \text{otherwise}
\end{cases}
\]

2. Draw the dataflow graph generated by the following program:

```c
for (int i = 0; i < 6; i++) {
    DFEVar x = io.input("x"+i, dfeInt(32));
    DFEVar y = x;
    if (i % 3 != 0) {
        for (int j = 0; j < 3; j++) {
            y = y + x*j;
        }
    } else {
        y = y * y;
    }
    io.output("y"+i, y, dfeInt(32));
}
```