The Last Mile to CAL Compiler for Epiphany Architecture

Mingkun Yang

HH

June 19th 2013

Outline

Table of Contents

Introduction

- Multilple-core trend
  - The end of higher frequency
  - Scale horizontally
- CAL Actor Language
  - DSL of dataflow program
  - Explicit parallelism

Brief intro to CAL

- Actor

Listing 1: Identify Actor

```
actor ID () In ==> Out :
    action In: [a] ==> Out: [a] end
end
```

- Network

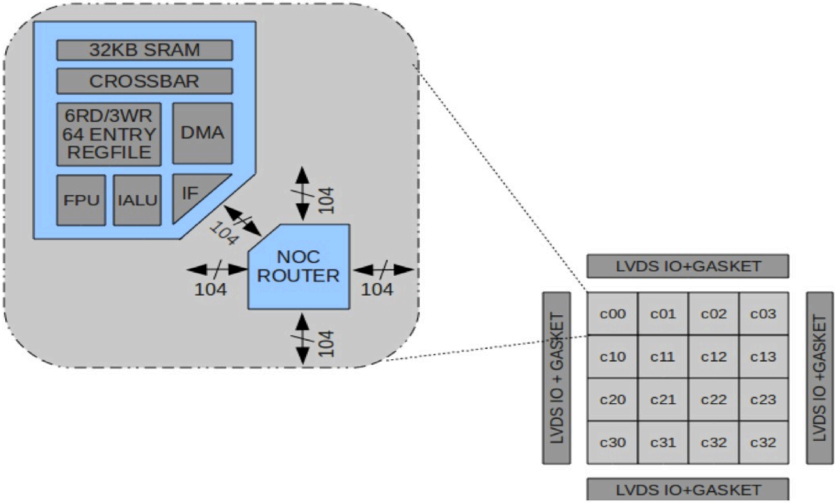
Listing 2: Simple network

```
x = ID();
y = ID();
...
x.out --> y.in;
...
```

Related work

- OpenDF
  - ACTORS project (d2c)[1]
- Orcc
  - Synthesis from Dataflow Programs[2]

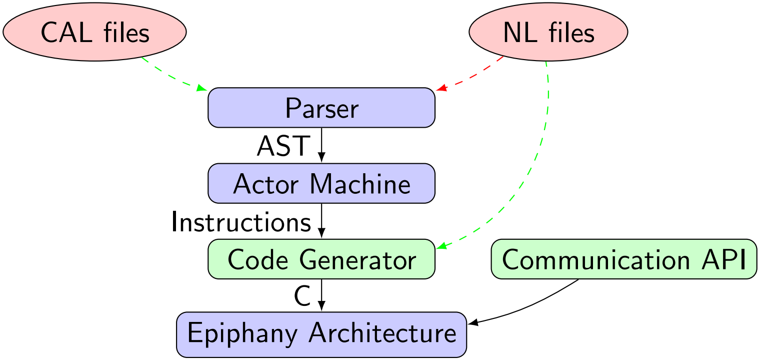
Epiphany Architecture



[3]

Table of Contents

CAL compiling process



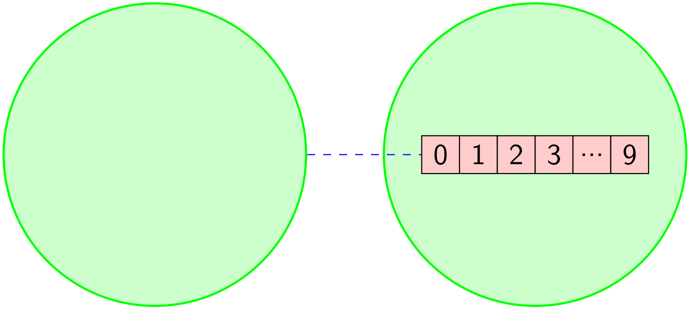
Code Generator

- Source-to-source compiler
  - Readability
  - Actor-based translation
    - One-to-one mapping (from .cal to .c)
    - Structure for one actor (Classes in OOP)
  - No duplication in linking
- Build Process
  - Eclipse (Default)
    - Each project for each core.
  - Customized Makefile
    - Stop on any failure from any core.

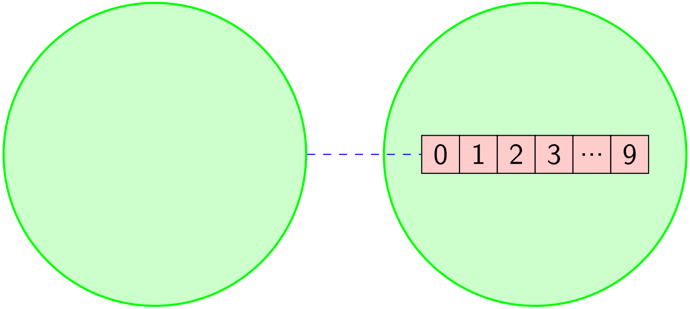
Communication API

- epiphany_write
- epiphany_read
- port_end
  - Called when there's no further tokens in this transaction.
  - Resembles end of packet in communication protocol.
  - Better measurement of active duration of each actor (each core).
- connect
  - Connect one input port with one output port.

Destination-buffer

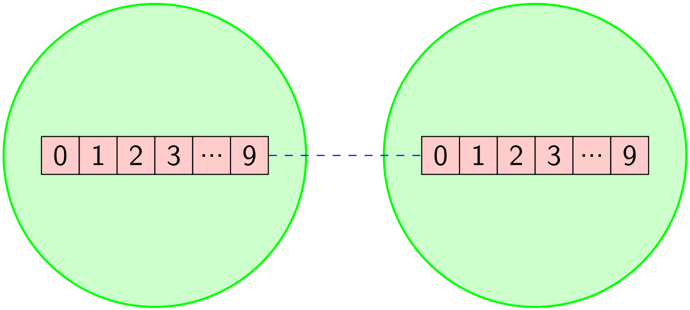


Destination-buffer

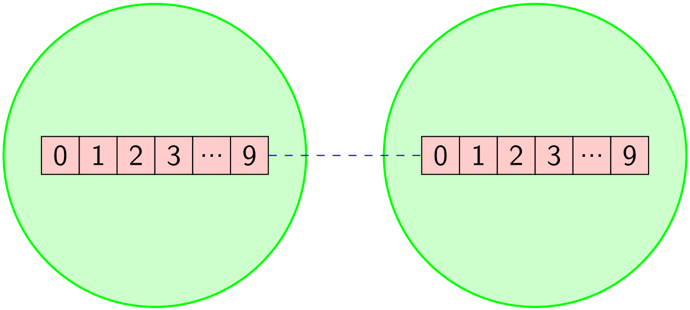


- Local > Remote

Both-buffer

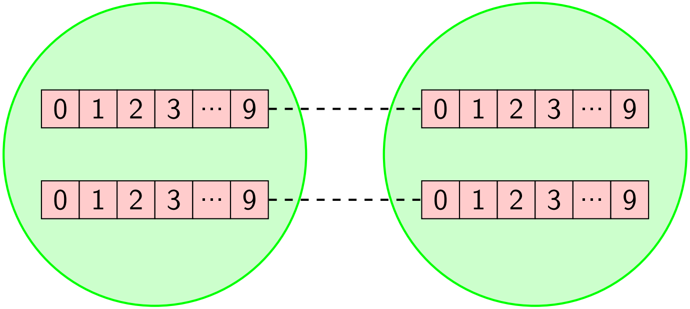


Both-buffer



- Direct Memory Access (DMA)
    - Separate “thread”

Double-buffer

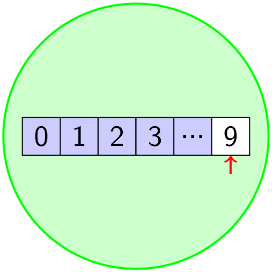


- Will be blocked if one is faster than the other

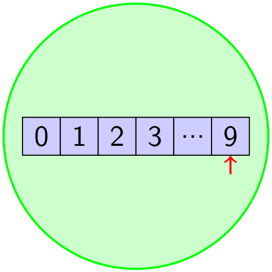
Design and Examples

- Design
  - Async by default (Never block)
  - Only block whene necessary (Fall back to sync call)
- Push tokens to the destination core(s).
  - try_flush vs do_flush

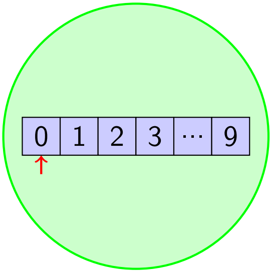
try_flush flow



(a) Initial condition



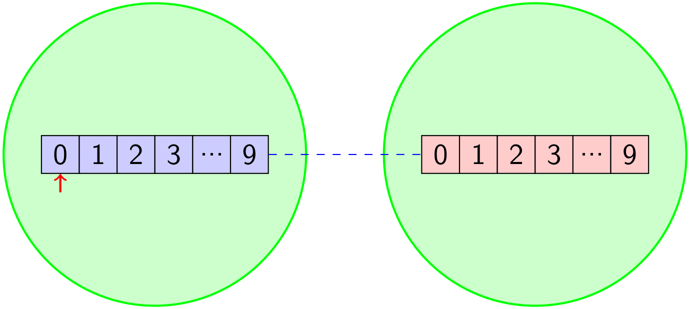
(b) Write to the last slot



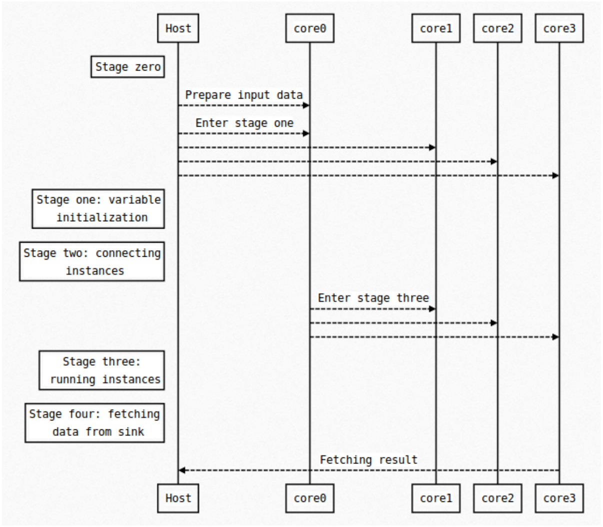
(c) Index pointer is updated

do_flush

- Red: unknown or uninterested
- Blue: occupied



Synchronization between the board and host


Host
core0
core1
core2
core3
Stage zero
Prepare input data
Enter stage one
Stage one: variable initialization
Stage two: connecting instances
Enter stage three
Stage three: running instances
Stage four: fetching data from sink
Fetching result
Host
core0
core1
core2
core3

Thread like management

- not_finished
  - There are further tokens from any of input ports.
  - Or there are tokens in the local buffer.
- run
  - Fire actions, consume and produce tokesn.
- end
  - Process all tokens in the local buffer.
  - Mark the end-of-token in this port.

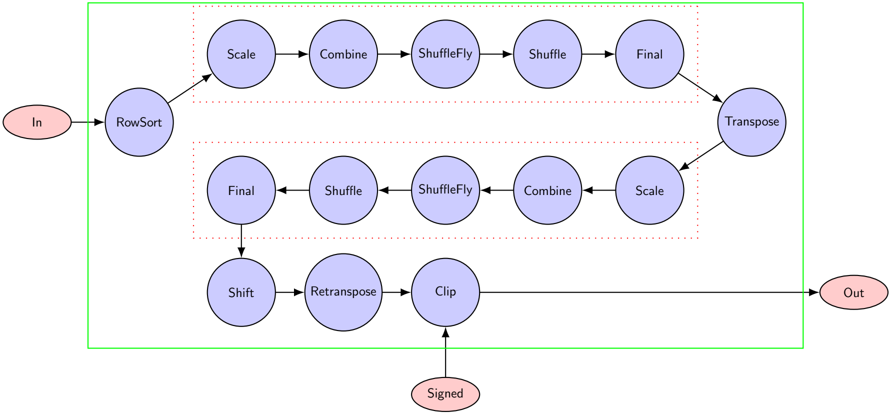
Strategy Pattern

Listing 3: How Actor Interface is Used

```
// common/common.c
...
void core_main(void *a, init_t *init) {
    ...
    while(api->not_finished(a)) {
        api->run(a);
    }
    api->end(a);
    ...
}
...
```

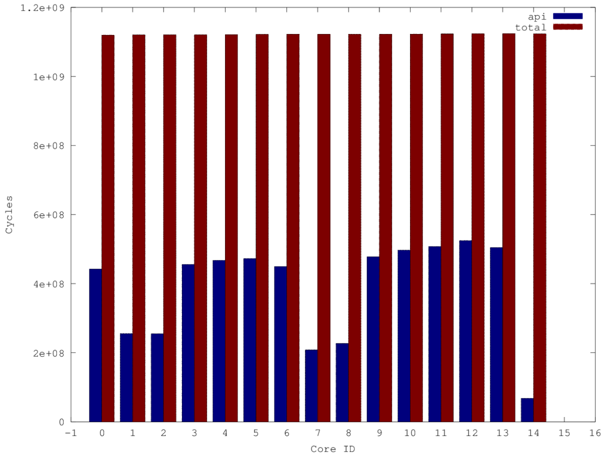
Table of Contents

IDCT2D

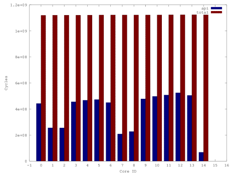


- 64000 tokens for *In* and 1000 tokens for *Signed*
- 100 times, and get the average

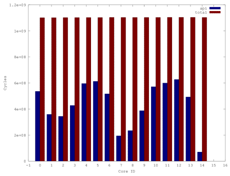
Destination-buffer



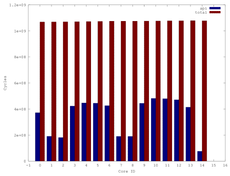
Compare three implementations of API



(d) Destination buffer



(e) Both-buffer



(f) Double-buffer

Max cycles of all cores

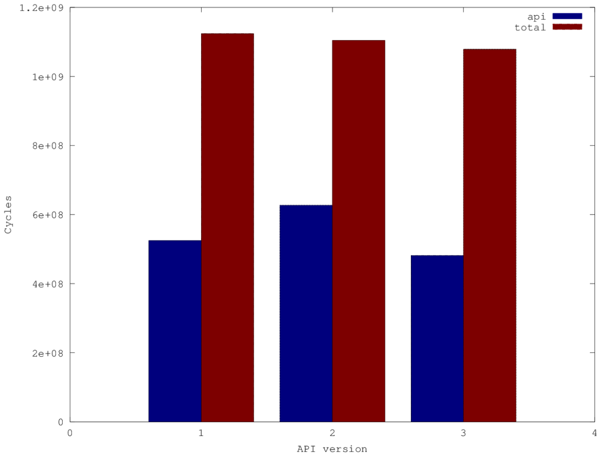


Table of Contents

On the Shoulder of Giants

- Front end
- Actor Machine
- Sequential C Code Generator
- The communication API for Epiphany

Resource

- http://www.actors-project.eu/
- Ghislain Roquier
  Hardware and Software Synthesis of Heterogeneous Systems from Dataflow Programs
- http://www.bdti.com/InsideDSP/2012/09/05/Adapteva
- https://bitbucket.org/albertnetymk/epiphany