The Last Mile to CAL Compiler for Epiphany Architecture

Mingkun Yang

HH

September 2, 2013

Outline

1. Introduction
2. Method
   - Incremental Refinement
   - Asynchronous and Synchronous Call
   - Synchronization
   - Manage all actors
3. Result
   - IDCT
   - Tiny buffer size
   - Medium buffer size
   - Large buffer size
4. Conclusion
5. Bonus: two mysterious bugs

Table of Contents

1. Introduction
2. Method
   - Incremental Refinement
   - Asynchronous and Synchronous Call
   - Synchronization
   - Manage all actors
3. Result
   - IDCT
   - Tiny buffer size
   - Medium buffer size
   - Large buffer size
4. Conclusion
5. Bonus: two mysterious bugs

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 : Identity Actor

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

- Network

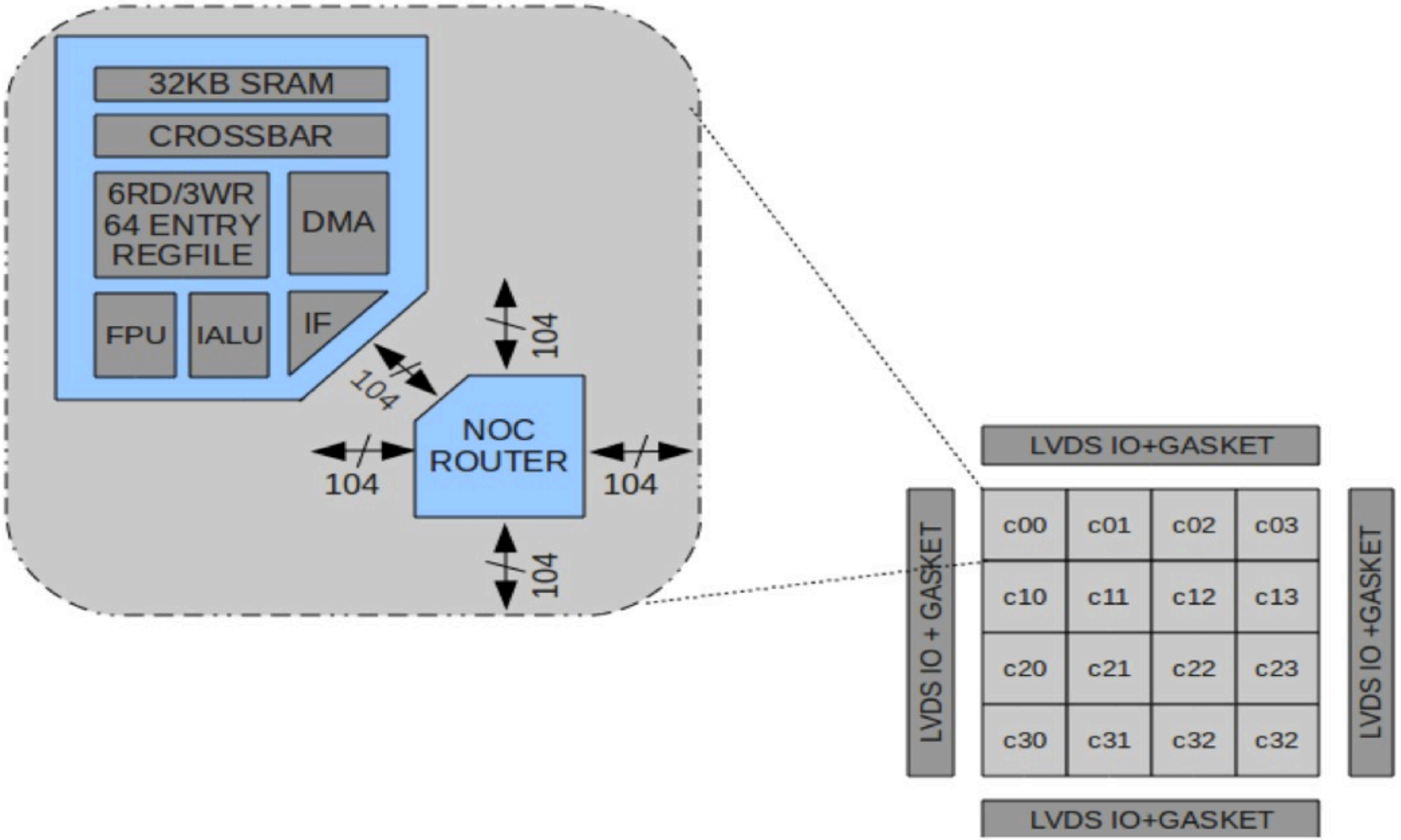
Listing 2 : Simple network

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

Related work

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

Epiphany Architecture

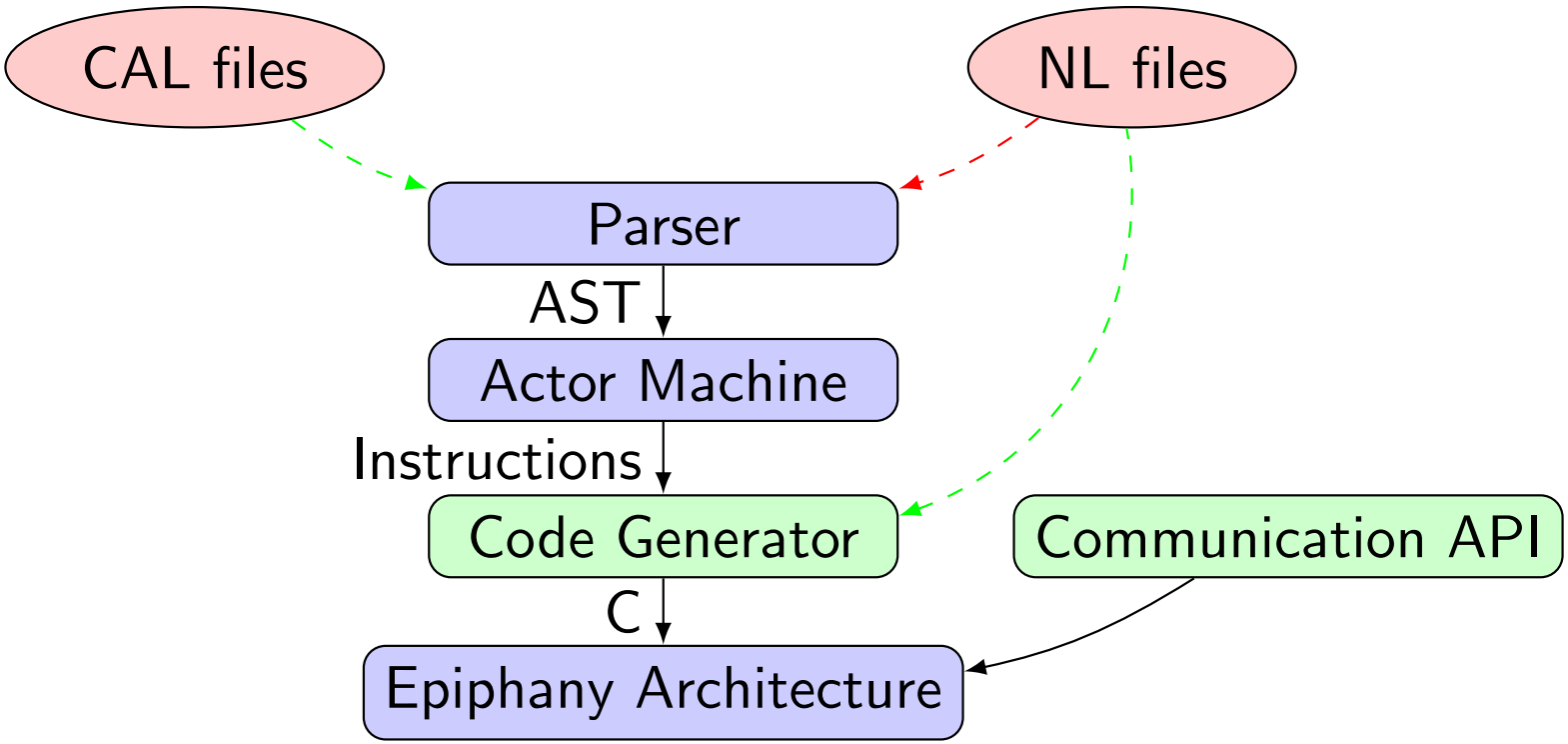


[3]

Table of Contents

1. Introduction
2. Method
   - Incremental Refinement
   - Asynchronous and Synchronous Call
   - Synchronization
   - Manage all actors
3. Result
   - IDCT
   - Tiny buffer size
   - Medium buffer size
   - Large buffer size
4. Conclusion
5. Bonus: two mysterious bugs

CAL compiling process



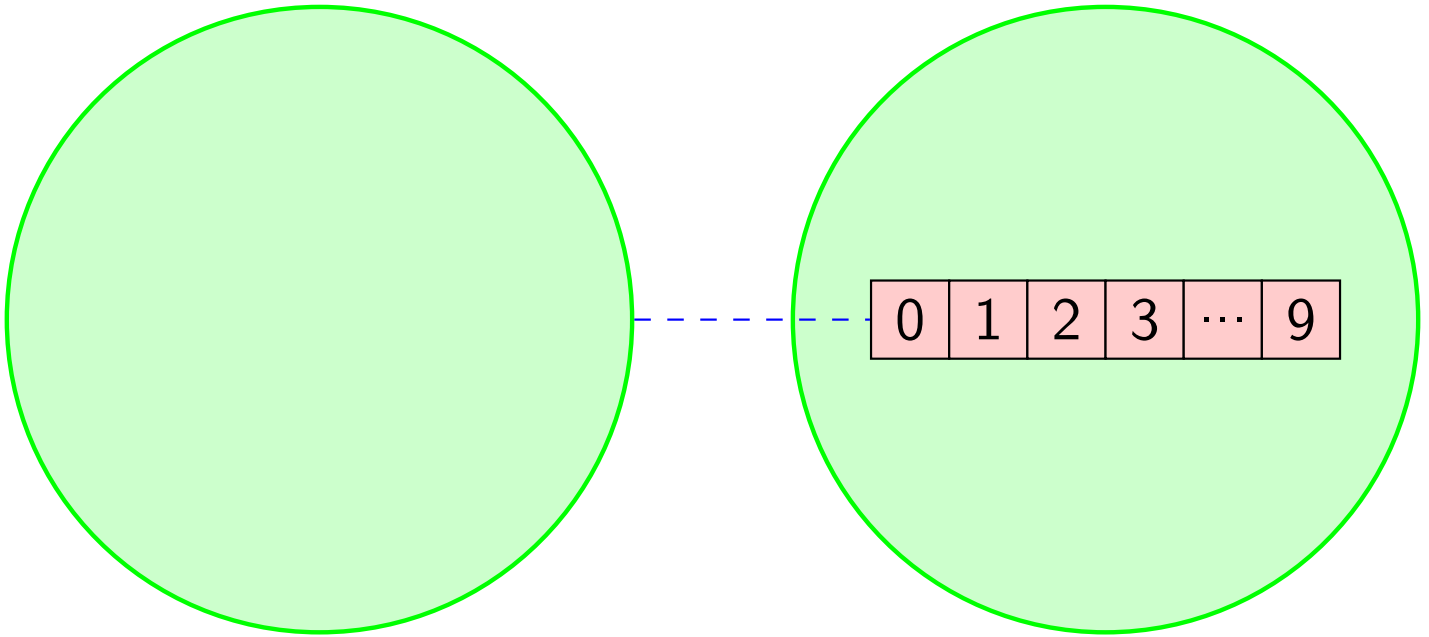
Code Generator

- 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.

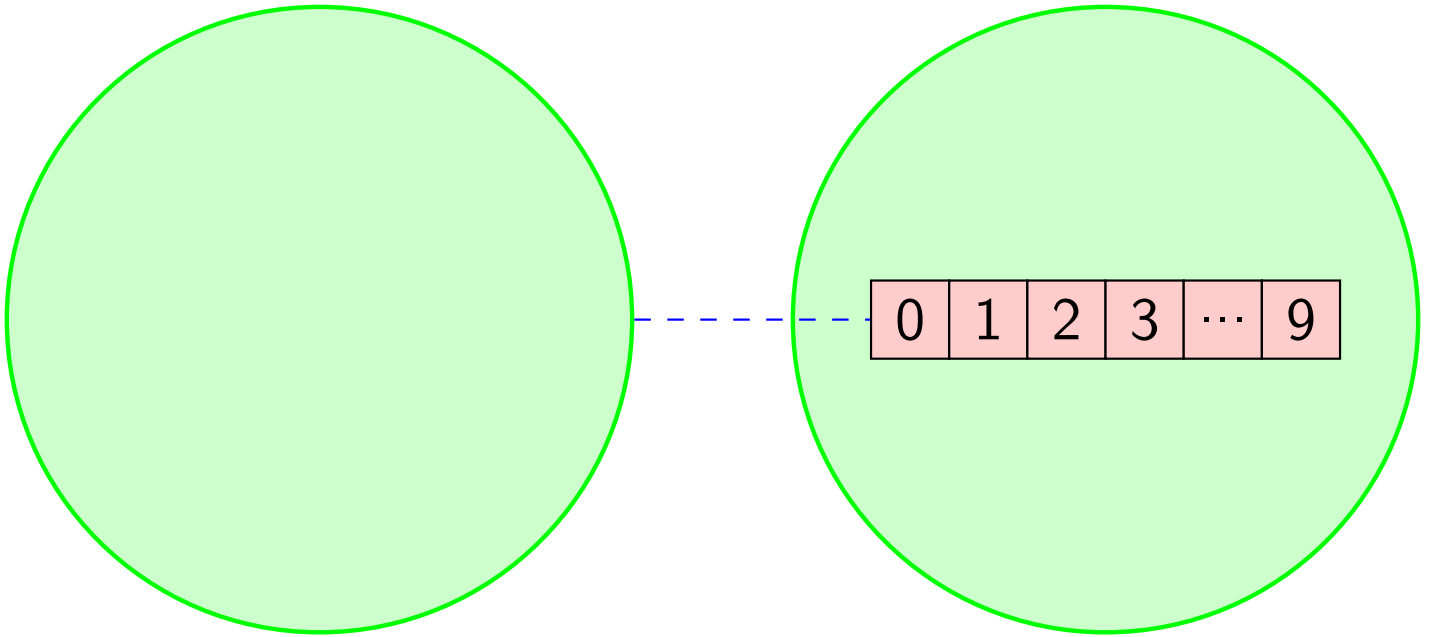
Communication API

- 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

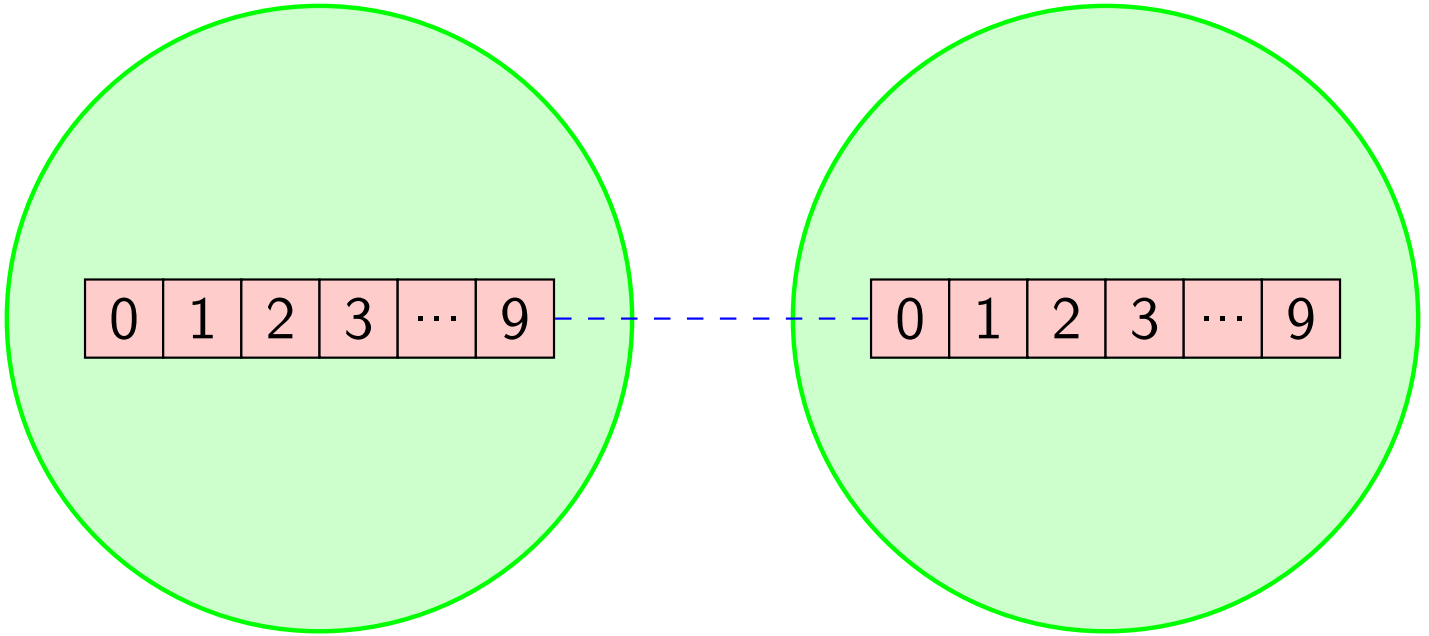


Destination-buffer

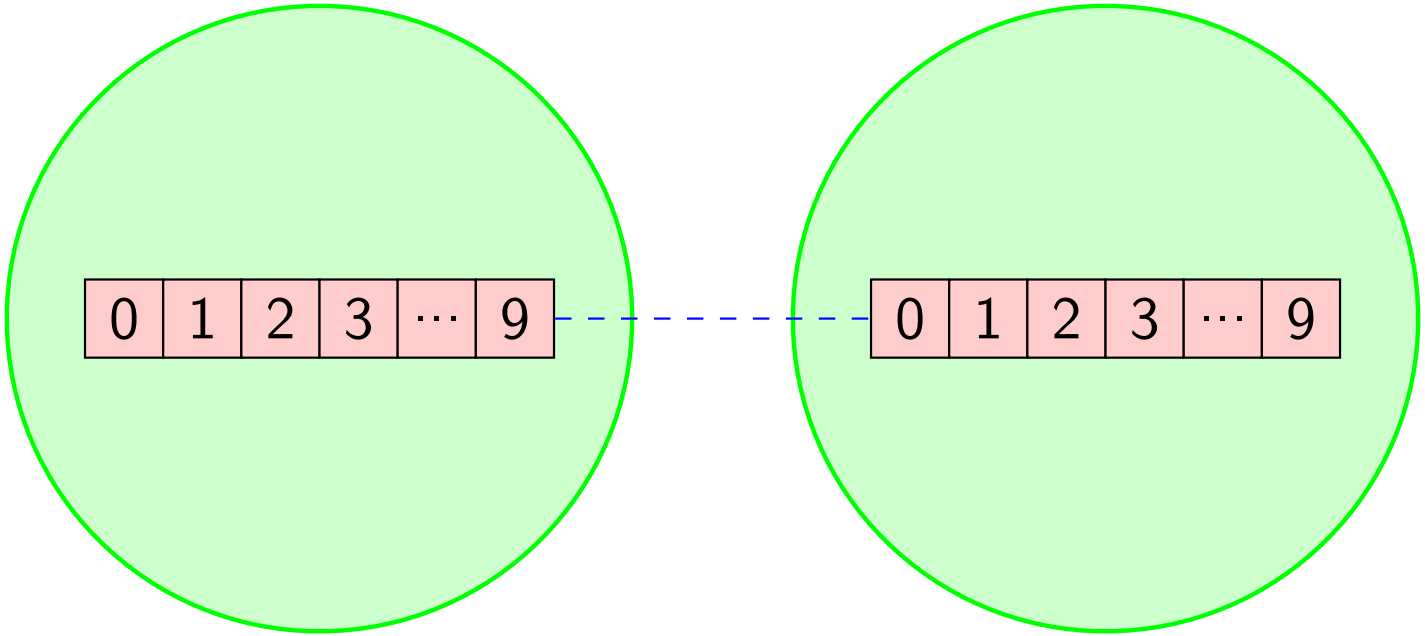


- Local > Remote

Both-buffer

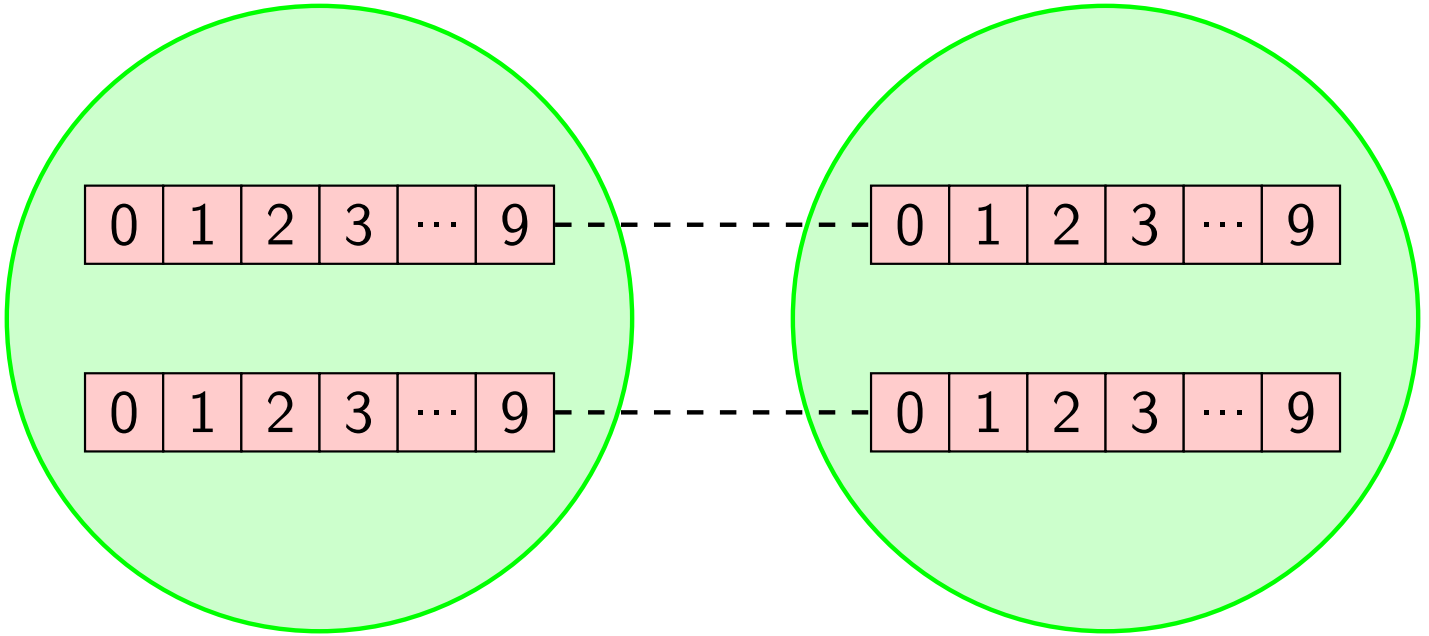


Both-buffer



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

Double-buffer

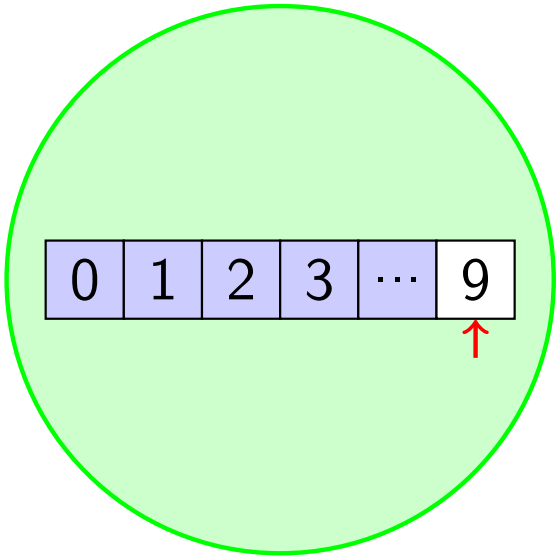


- Will be blocked if one is faster than the other

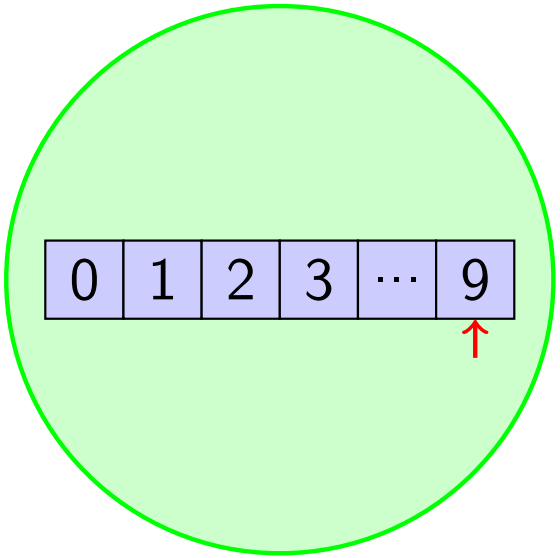
Design and Examples

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

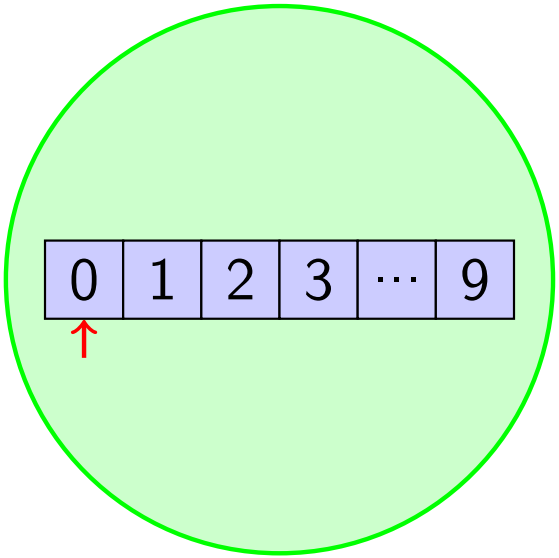
try_flush flow



(a) Initial condition



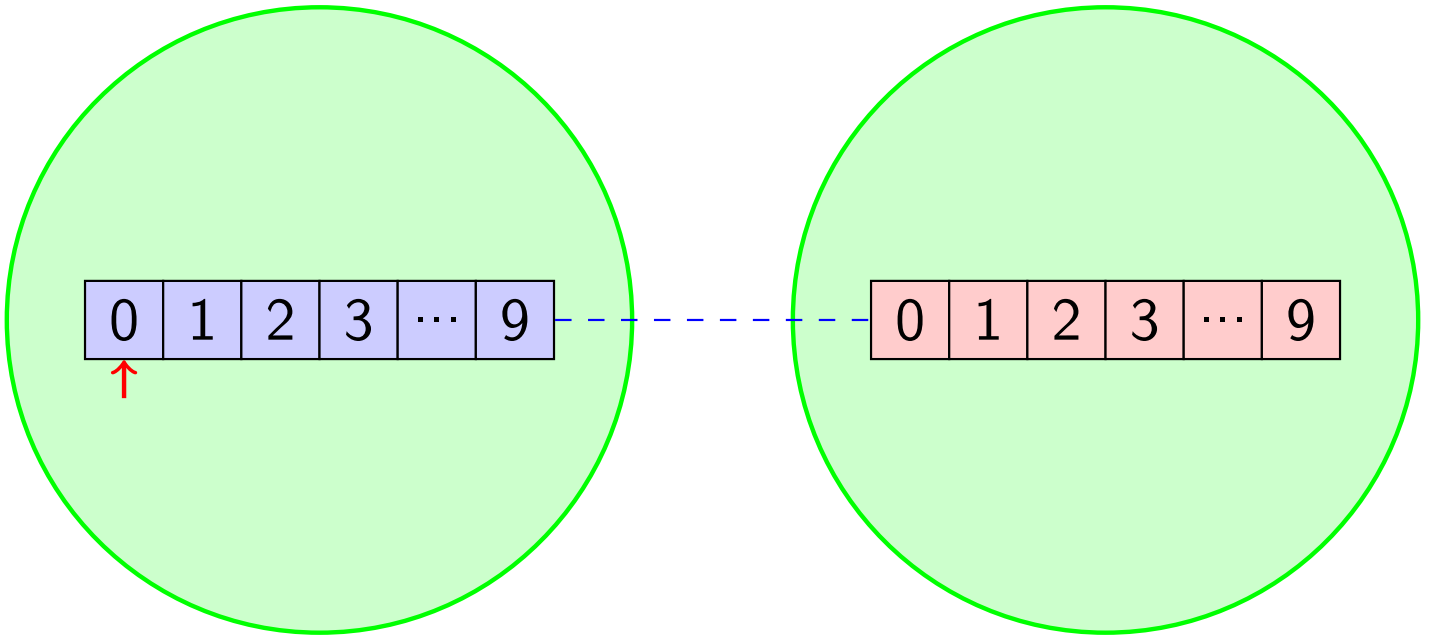
(b) Write to the last slot



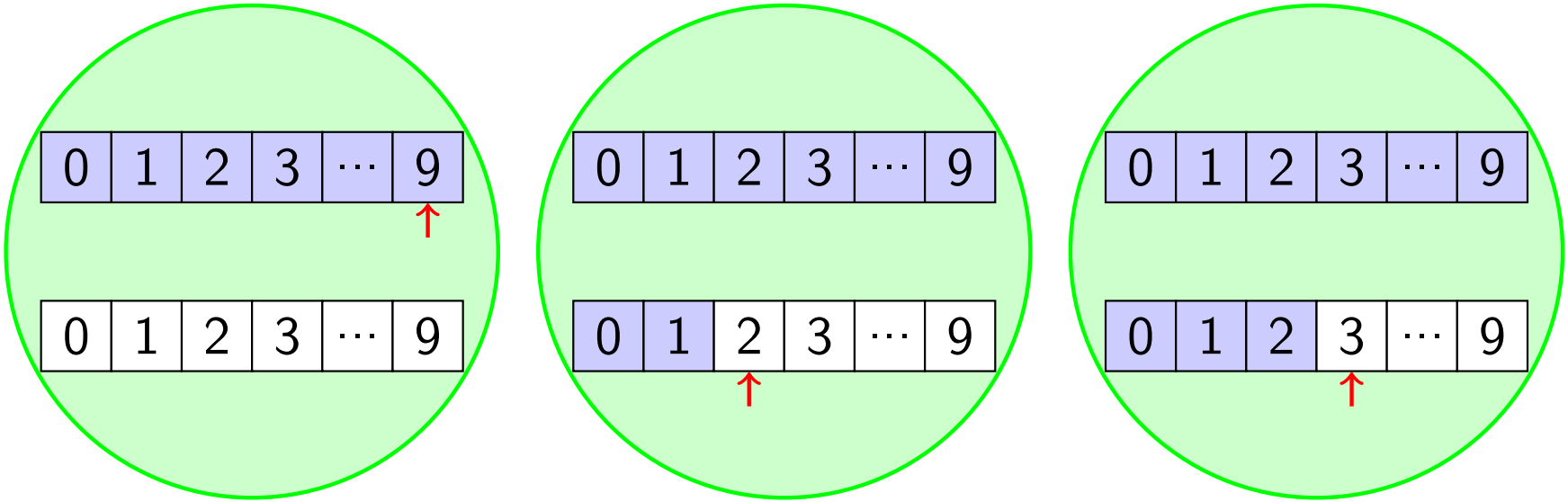
(c) Index pointer is updated

do_flush

- Red: unknown or uninterested
- Blue: occupied

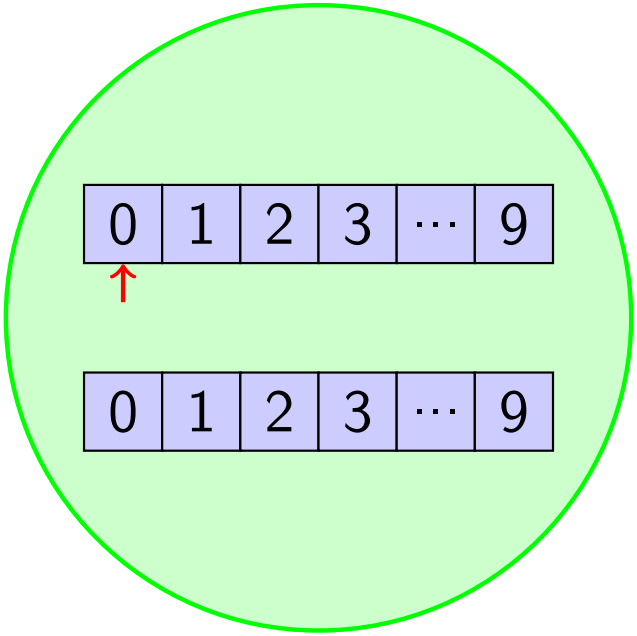


try_distribute

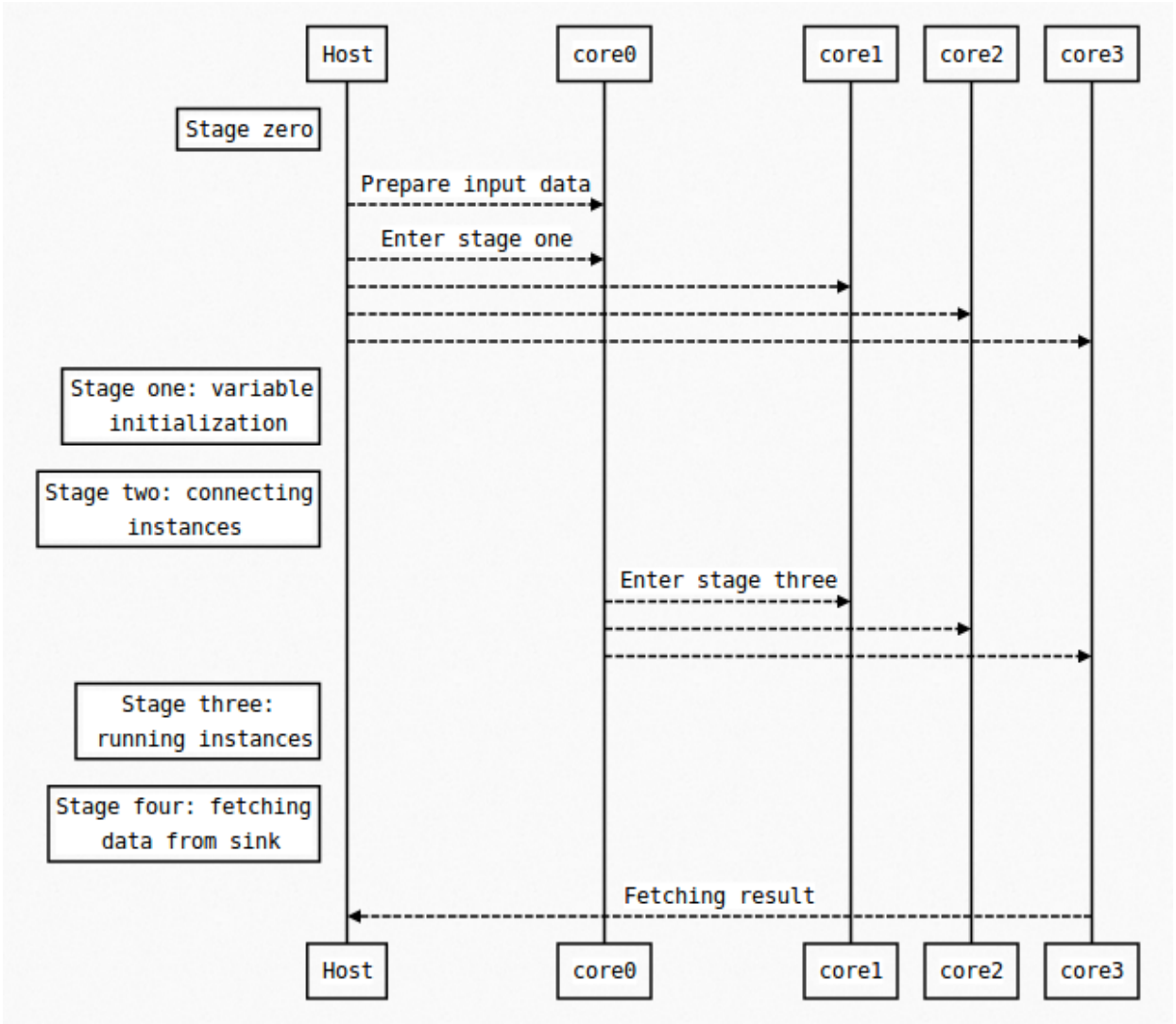


(a) One fifo becomes full (b) Write when one fifo is full (c) Another write operation

do_distribute



Synchronization between the board and host



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

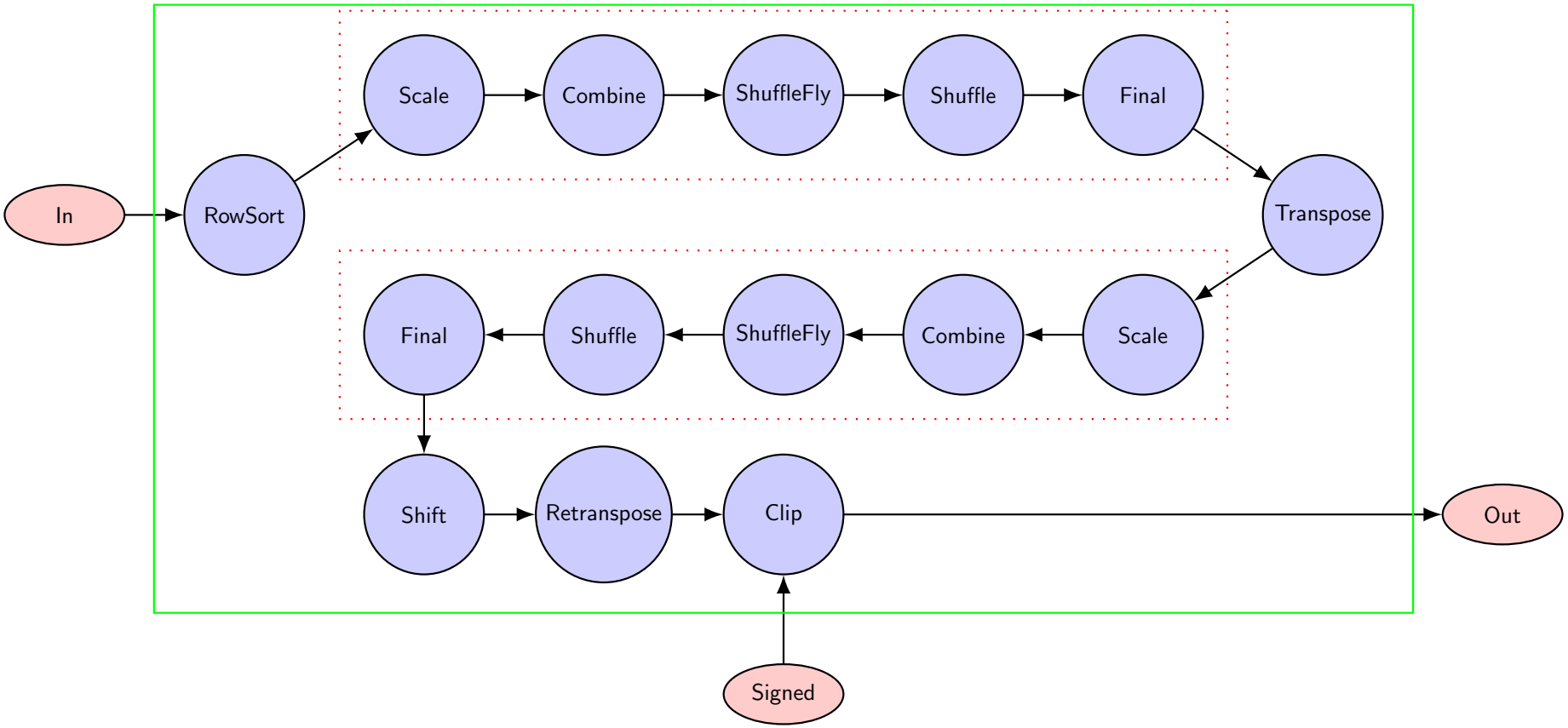
Listing 3 : How Actor Interface is Used

```
// 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

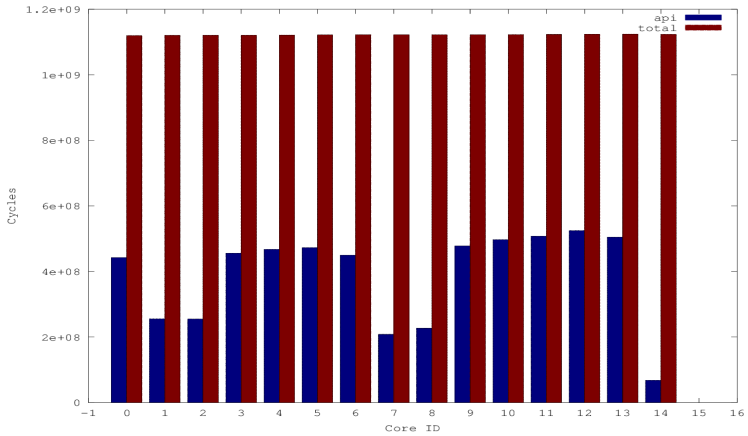
1. Introduction
2. Method
   - Incremental Refinement
   - Asynchronous and Synchronous Call
   - Synchronization
   - Manage all actors
3. Result
   - IDCT
   - Tiny buffer size
   - Medium buffer size
   - Large buffer size
4. Conclusion
5. Bonus: two mysterious bugs

IDCT2D

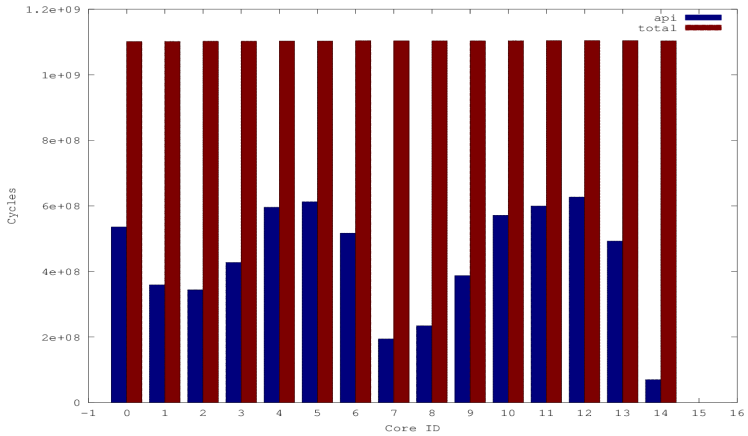


- 64000 tokens for *In* and 1000 tokens for *Signed*
- 100 times, and get the average

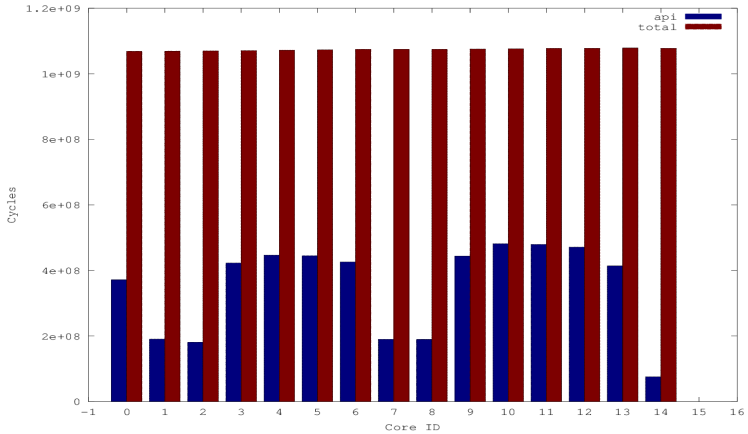
Tiny buffer size: api call vs total execution



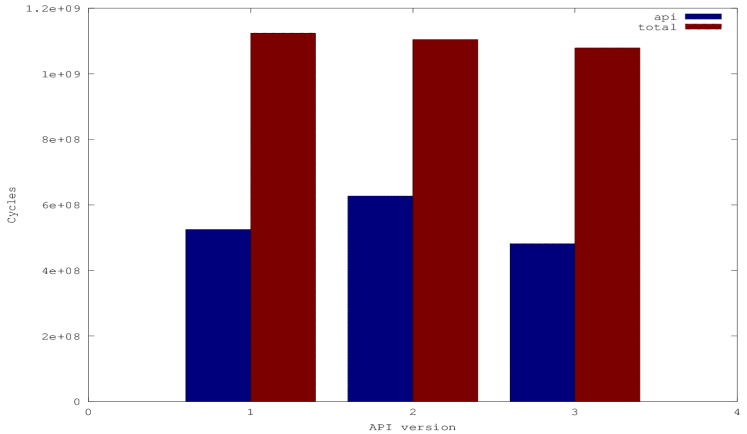
(a) destination buffer version



(b) both buffer version

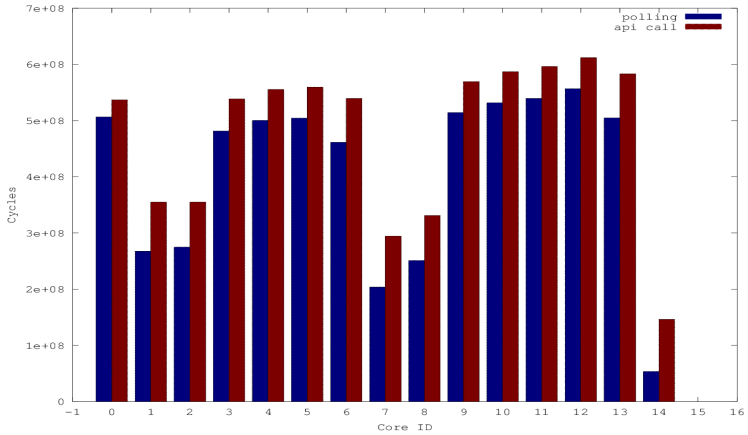


(c) double buffer version

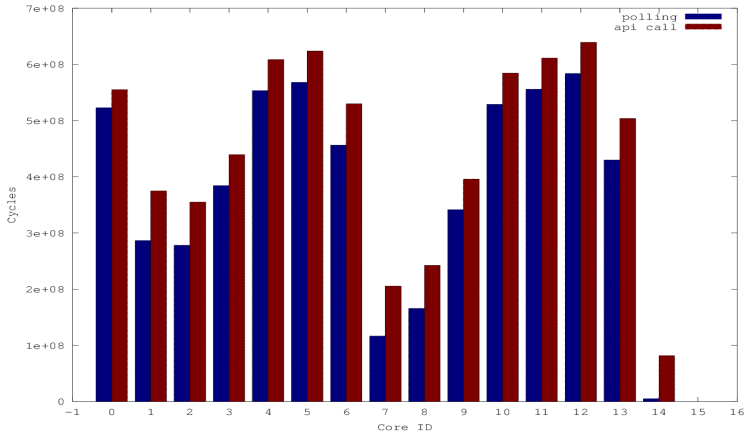


(d) maximum from all three version

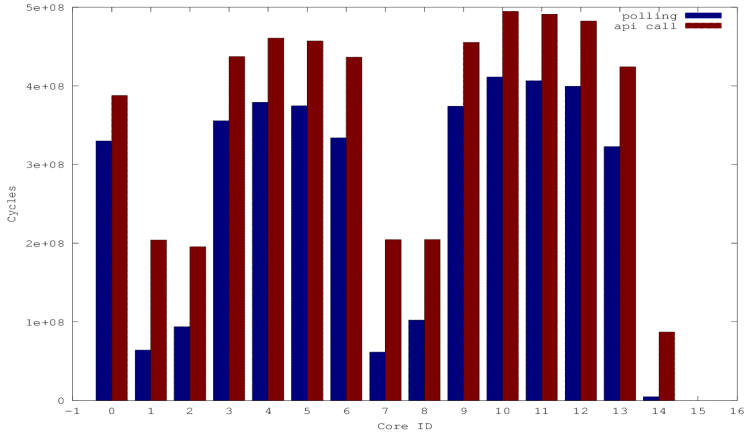
Tiny buffer size: polling vs api call



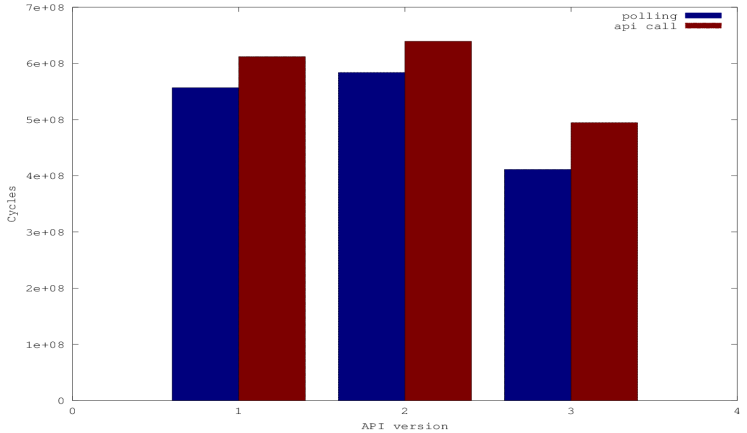
(a) destination buffer version



(b) both buffer version

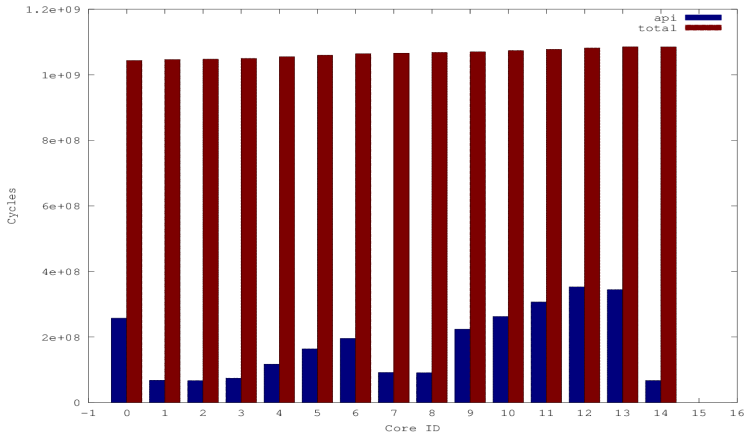


(c) double buffer version

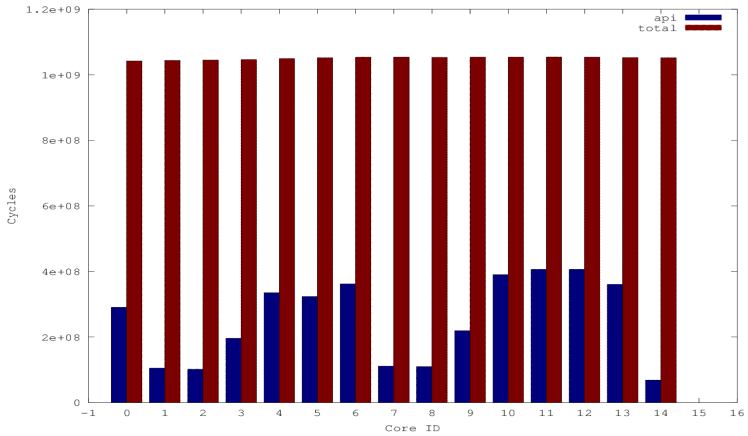


(d) maximum from all three version

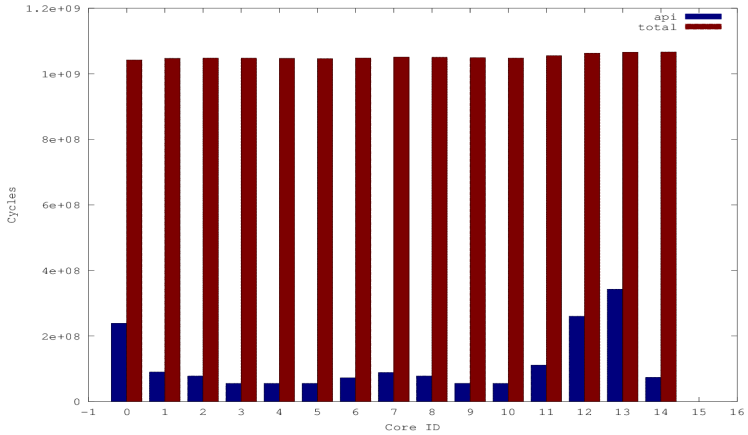
Medium buffer size: api call vs total execution



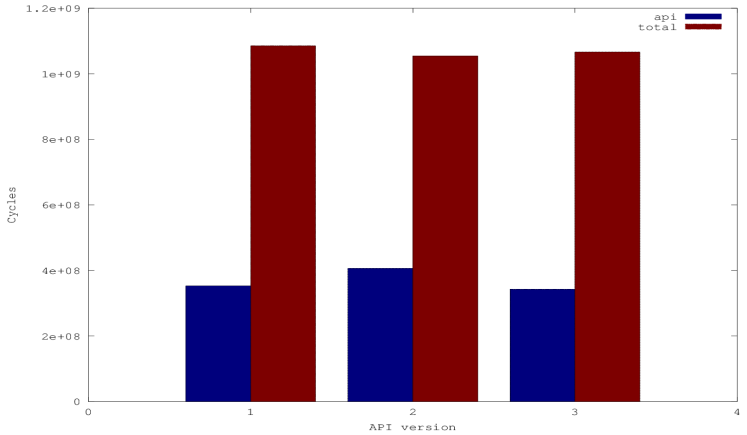
(a) destination buffer version



(b) both buffer version

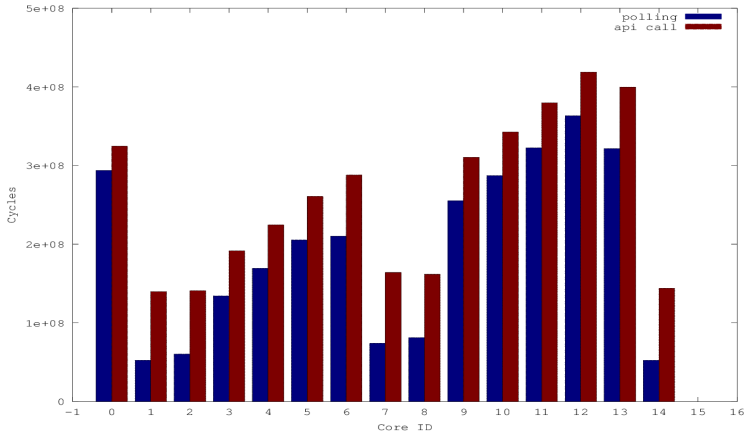


(c) double buffer version

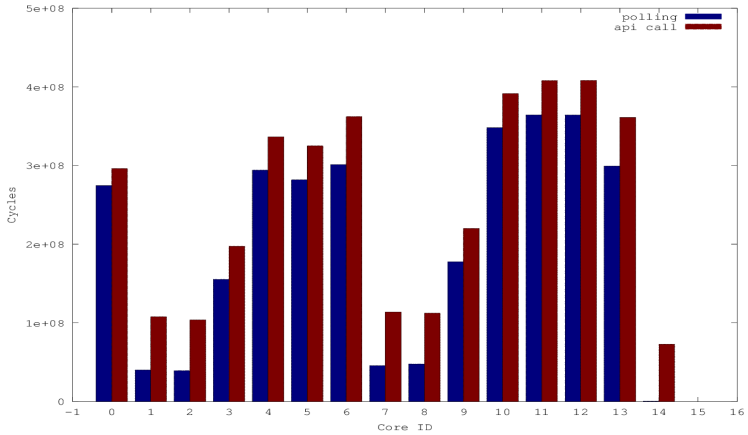


(d) maximum from all three version

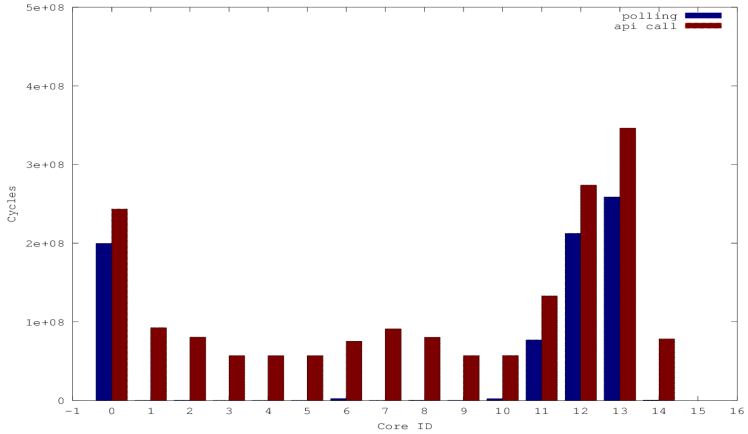
Medium buffer size: polling vs api call



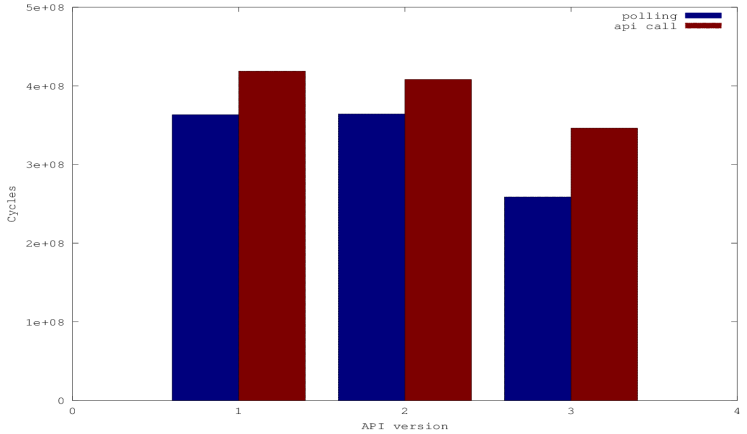
(a) destination buffer version



(b) both buffer version

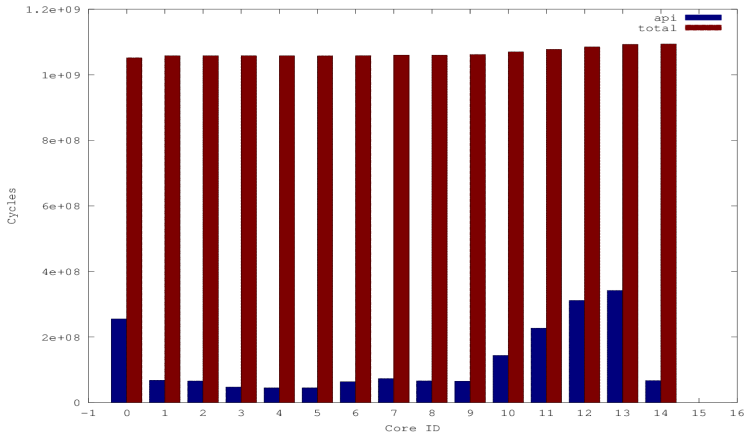


(c) double buffer version

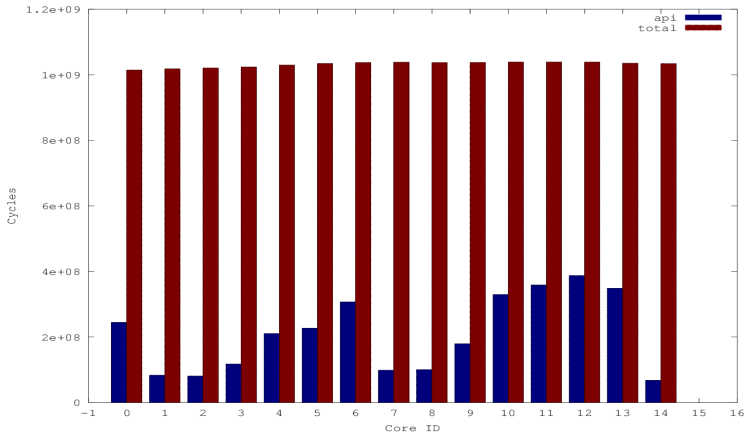


(d) maximum from all three version

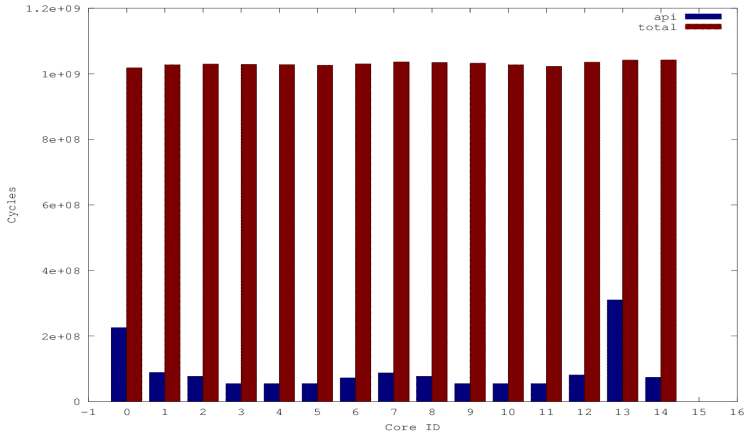
Large buffer size: api call vs total execution



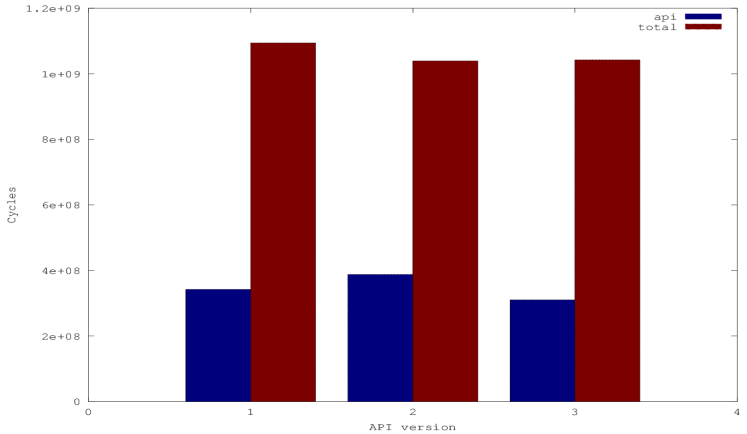
(a) destination buffer version



(b) both buffer version

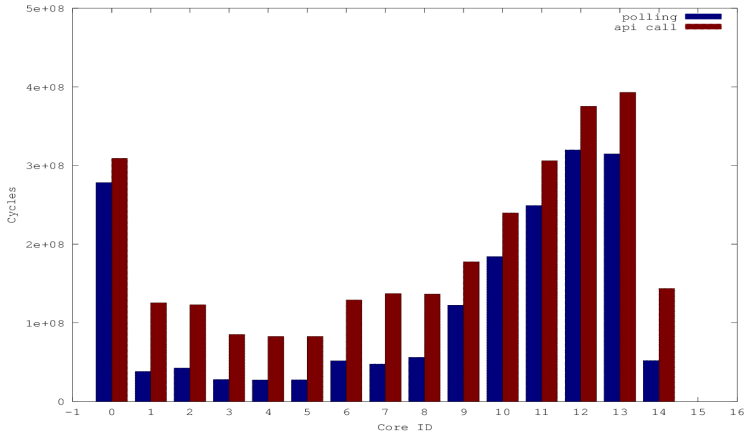


(c) double buffer version

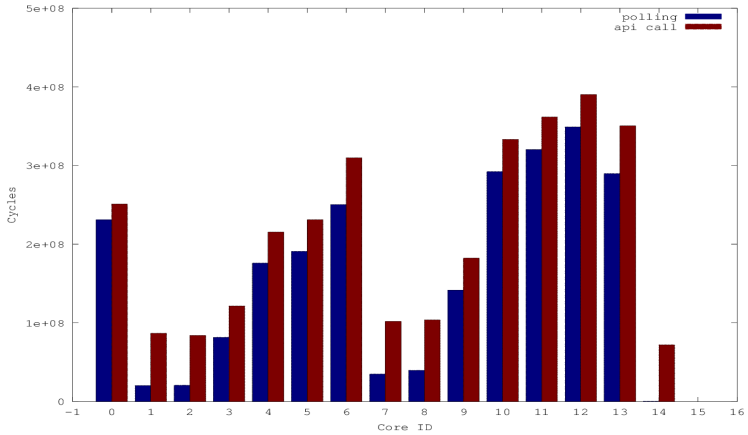


(d) maximum from all three version

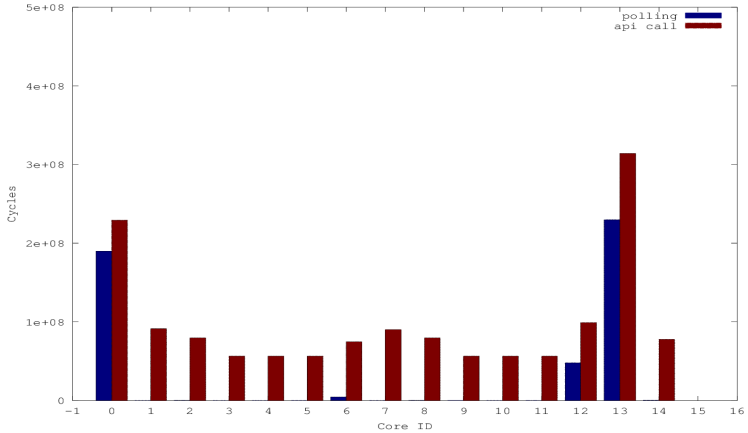
Large buffer size: polling vs api call



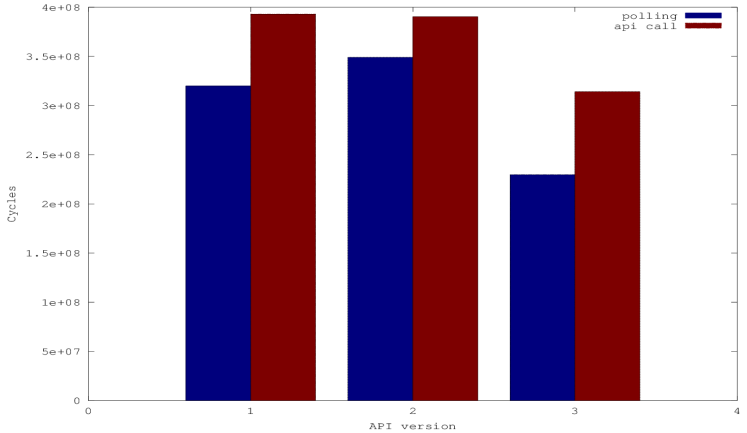
(a) destination buffer version



(b) both buffer version



(c) double buffer version



(d) maximum from all three version

Table of Contents

1. Introduction
2. Method
   - Incremental Refinement
   - Asynchronous and Synchronous Call
   - Synchronization
   - Manage all actors
3. Result
   - IDCT
   - Tiny buffer size
   - Medium buffer size
   - Large buffer size
4. Conclusion
5. Bonus: two mysterious bugs

On the Shoulder of Giants

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

Conclusion and future work

- Actor is lightweight, and it's better to assign multiple actors to one core.
  - Actor composition in Actor Machine
  - multiple actors in one core – user space thread

Conclusion and future work

- Actor is lightweight, and it's better to assign multiple actors to one core.
  - Actor composition in Actor Machine
  - multiple actors in one core – user space thread
- Balance the effort between creating tests and extra caution from developers
  - Create reasonable amount of unit tests.
  - Be focused while developing.

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

Table of Contents

1. Introduction
2. Method
   - Incremental Refinement
   - Asynchronous and Synchronous Call
   - Synchronization
   - Manage all actors
3. Result
   - IDCT
   - Tiny buffer size
   - Medium buffer size
   - Large buffer size
4. Conclusion
5. Bonus: two mysterious bugs

Listing 4 : buggy code

```
typedef struct {

    int players;
    int source[20];
    ...
}
```

Listing 5 : fix

```
typedef struct {
    int dummy;
    int players;
    int source[20];
    ...
}
```

Listing 4 : buggy code

```
typedef struct {

    int players;
    int source[20];
    ...
}
```

Listing 5 : fix

```
typedef struct {
    int dummy;
    int players;
    int source[20];
    ...
}
```

- Observation

Listing 4 : buggy code

```
typedef struct {

    int players;
    int source[20];
    ...
}
```

Listing 5 : fix

```
typedef struct {
    int dummy;
    int players;
    int source[20];
    ...
}
```

- Observation
  - Consistent and deterministic.

Listing 4 : buggy code

```
typedef struct {

    int players;
    int source[20];
    ...
}
```

Listing 5 : fix

```
typedef struct {
    int dummy;
    int players;
    int source[20];
    ...
}
```

- Observation
  - Consistent and deterministic.
  - 64 bit host

Listing 4 : buggy code

```
typedef struct {

    int players;
    int source[20];
    ...
}
```

Listing 5 : fix

```
typedef struct {
    int dummy;
    int players;
    int source[20];
    ...
}
```

- Observation
  - Consistent and deterministic.
  - 64 bit host
  - sizeof(int) == 4

Listing 4 : buggy code

```
typedef struct {

    int players;
    int source[20];
    ...
}
```

Listing 5 : fix

```
typedef struct {
    int dummy;
    int players;
    int source[20];
    ...
}
```

- Observation
  - Consistent and deterministic.
  - 64 bit host
  - sizeof(int) == 4
- Explanation
  - field alignment in structure

Listing 6 : buggy code

```
bool might_has_input(port_in *p)
{
    return has_input(p, 1) || !p->end ;
}
```

Listing 7 : fix

```
bool might_has_input(port_in *p)
{
    return !p->end || has_input(p, 1) ;
}
```

- Observation

- Observation
  - Non-deterministic.

- Observation
  - Non-deterministic.
  - precedence, assembly, ...

- Observation
  - Non-deterministic.
  - precedence, assembly, ...
- Explanation (with external help)
  - Shared resource is modified between two checks.
  - Check them in the reverse order of mutating.