Argonne
NATIONAL LABORATORY

EXTENDING CODELET MODEL FOR DATAFLOW SOFTWARE PIPELINING USING SOFTWARE-HARDWARE CO-DESIGN

Siddhisanket Raskar, Thomas Applencourt, Kalyan Kumaran, Guang Gao

SIDDHISANKET RASKAR
Research Aide, Argonne National Lab.
PhD Candidate, University of Delaware.

19 July 2019
Milwaukee, Wisconsin.

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Outline

Background & Motivation

- Software Pipelining
- Dataflow Software Pipelining
- Codelet Model

Solution Methodology

- Extension of Codelet PXM
- Extension of CAM
- Optimizations

Problem Formulation

- Defining Class of Codelet Graph
- Challenges

Future Work

- Implementation of PXM & CAM
- Kernels & Applications

Outline

Background & Motivation

- Software Pipelining
- Dataflow Software Pipelining
- Codelet Model

Solution Methodology

- Extension of Codelet PXM
- Extension of CAM
- Optimizations

Problem Formulation

- Defining Class of Codelet Graph
- Challenges

Future Work

- Implementation of PXM & CAM
- Kernels & Applications

Software Pipelining

Basics

Without Software Pipelining

Execution of loop without Software Pipelining -

- Each iteration takes 9 cycles to produce result.
- The next iteration starts after the earlier iteration finishes.
- Loop unrolling techniques can be used.

```
L  :  for (i=0 ; I < 3 ; i++)

s1:      a[i] = a[i] + 1 ;
s2:      b[i] = a[i] + 1 ;
s3:      c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There is only 1 compute unit for each instruction in pipeline.
- Lets assume each instruction takes 3 cycles

Cycle	I1	I2	I3
1	s1		
2			
3			
4	s2		
5			
6			
7	s3		
8			
9			
10		s1	
11			
12			
13		s2	
14			
15			
16		s3	
⋮			

Schedule with no Software Pipelining

Software Pipelining

Basics

With Software Pipelining

Is the technique used for loop optimization where iteration of a loop is activated before its preceding iteration is complete.

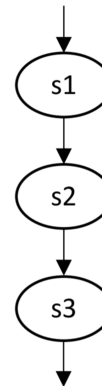
- Instructions are reordered as long as they satisfy the dependencies.
- It's a static, compile time technique.
- It uses dependence graph to compute schedule statically.

```
L  :  for (i=0 ; I < 3 ; i++)

s1:      a[i] = a[i] + 1 ;
s2:      b[i] = a[i] + 1 ;
s3:      c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There is only 1 functional unit for each task in pipeline.
- Lets assume each instruction takes 3 cycles



Dependence Graph

Cycle	I1	I2	I3
1	s1		
2		s1	
3			s1
4	s2		
5		s2	
6			s2
7	s3		
8		s3	
9			s3
10			
11			
⋮			

Schedule with Software Pipelining

Software Pipelining
Advantages

Advantages

- Throughput of the system is improved with schedule with software pipelining.

Rau et al.81 (MICRO14)
Modulo Scheduling

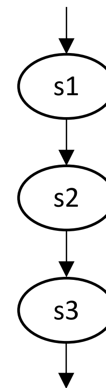
Rau et al.94 (MICRO27)
Iterative Modulo Scheduling

```
L  :  for (i=0 ; I < 3 ; i++)

s1:      a[i] = a[i] + 1 ;
s2:      b[i] = a[i] + 1 ;
s3:      c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There is only 1 functional unit for each task in pipeline.
- Lets assume each instruction takes 3 cycles



Dependence Graph

Cycle	I1	I2	I3
1	s1		
2		s1	
3			s1
4	s2		
5		s2	
6			s2
7	s3		
8		s3	
9			s3
10			
11			
⋮			

Schedule with Software Pipelining

Cycle	I1	I2	I3
1	s1		
2			
3			
4	s2		
5			
6			
7	s3		
8			
9			
10		s1	
11			
12			
13		s2	
14			
15			
16		s3	
⋮			

Schedule with no Software Pipelining

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

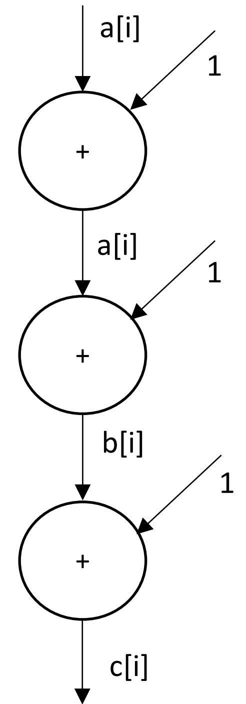
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L   :   for (i=0 ; i < 3 ; i++)

s1:       a[i] = a[i] + 1 ;
s2:       b[i] = a[i] + 1 ;
s3:       c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.



Only Loop Body

	Tokens Iteration 1
	Tokens Iteration 2
	Tokens Iteration 3

G. R. Gao, “Algorithmic aspects of balancing techniques for pipelined data flow code generation,” *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

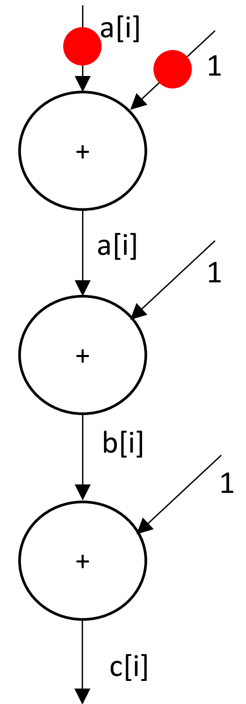
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L   :  for (i=0 ; i < 3 ; i++)

s1:     a[i] = a[i] + 1 ;
s2:     b[i] = a[i] + 1 ;
s3:     c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.



Only Loop Body

(red)	Tokens Iteration 1
(blue)	Tokens Iteration 2
(green)	Tokens Iteration 3

G. R. Gao, "Algorithmic aspects of balancing techniques for pipelined data flow code generation," *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

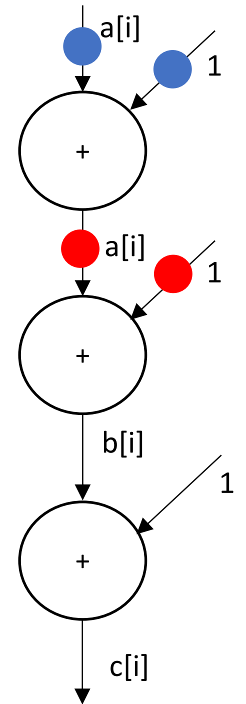
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L  :  for (i=0 ; i < 3 ; i++)

s1:     a[i] = a[i] + 1 ;
s2:     b[i] = a[i] + 1 ;
s3:     c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.



Only Loop Body

(red)	Tokens Iteration 1
(blue)	Tokens Iteration 2
(green)	Tokens Iteration 3

G. R. Gao, “Algorithmic aspects of balancing techniques for pipelined data flow code generation,” *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

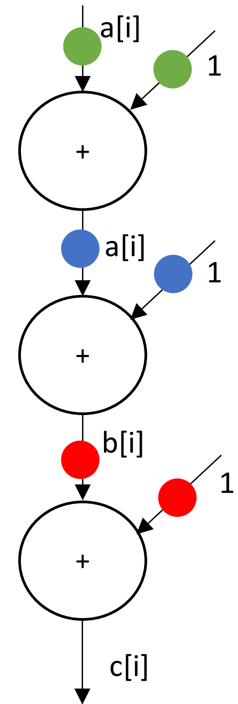
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L  :  for (i=0 ; i < 3 ; i++)

s1:      a[i] = a[i] + 1 ;
s2:      b[i] = a[i] + 1 ;
s3:      c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.



Only Loop Body

(red)	Tokens Iteration 1
(blue)	Tokens Iteration 2
(green)	Tokens Iteration 3

G. R. Gao, "Algorithmic aspects of balancing techniques for pipelined data flow code generation," *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

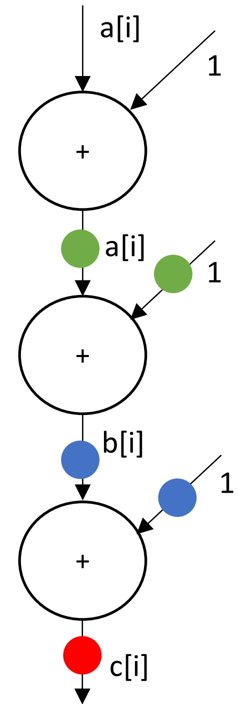
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L   :  for (i=0 ; i < 3 ; i++)

s1:       a[i] = a[i] + 1 ;
s2:       b[i] = a[i] + 1 ;
s3:       c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.



Only Loop Body

(red)	Tokens Iteration 1
(blue)	Tokens Iteration 2
(green)	Tokens Iteration 3

G. R. Gao, “Algorithmic aspects of balancing techniques for pipelined data flow code generation,” *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

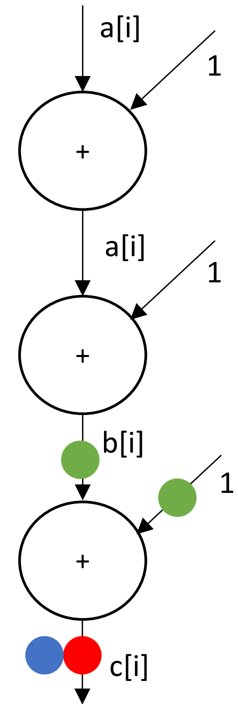
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L   :  for (i=0 ; i < 3 ; i++)

s1:      a[i] = a[i] + 1 ;
s2:      b[i] = a[i] + 1 ;
s3:      c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.



Only Loop Body

(red)	Tokens Iteration 1
(blue)	Tokens Iteration 2
(green)	Tokens Iteration 3

G. R. Gao, "Algorithmic aspects of balancing techniques for pipelined data flow code generation," *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Dataflow Software Pipelining

Basics

Dataflow Software Pipelining

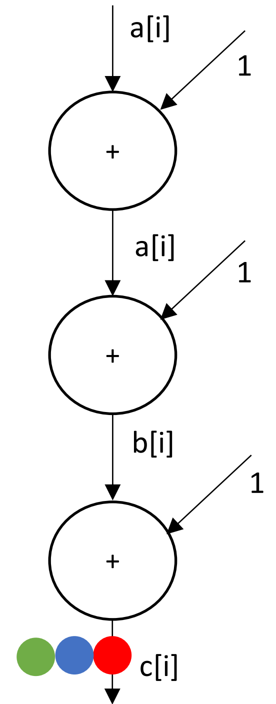
- Dataflow Software Pipelining is compile time as well as runtime technique.
- The naturally available information about dependencies is used at runtime for scheduling.
- Tokens from various iterations of the loop are executed.

```
L   :  for (i=0 ; i < 3 ; i++)

s1:      a[i] = a[i] + 1 ;
s2:      b[i] = a[i] + 1 ;
s3:      c[i] = b[i] + 1 ;
```

C++ Code : Simple Loop

Assumption

- There are 3 PEs.
- Lets assume each instruction takes 1 cycle.

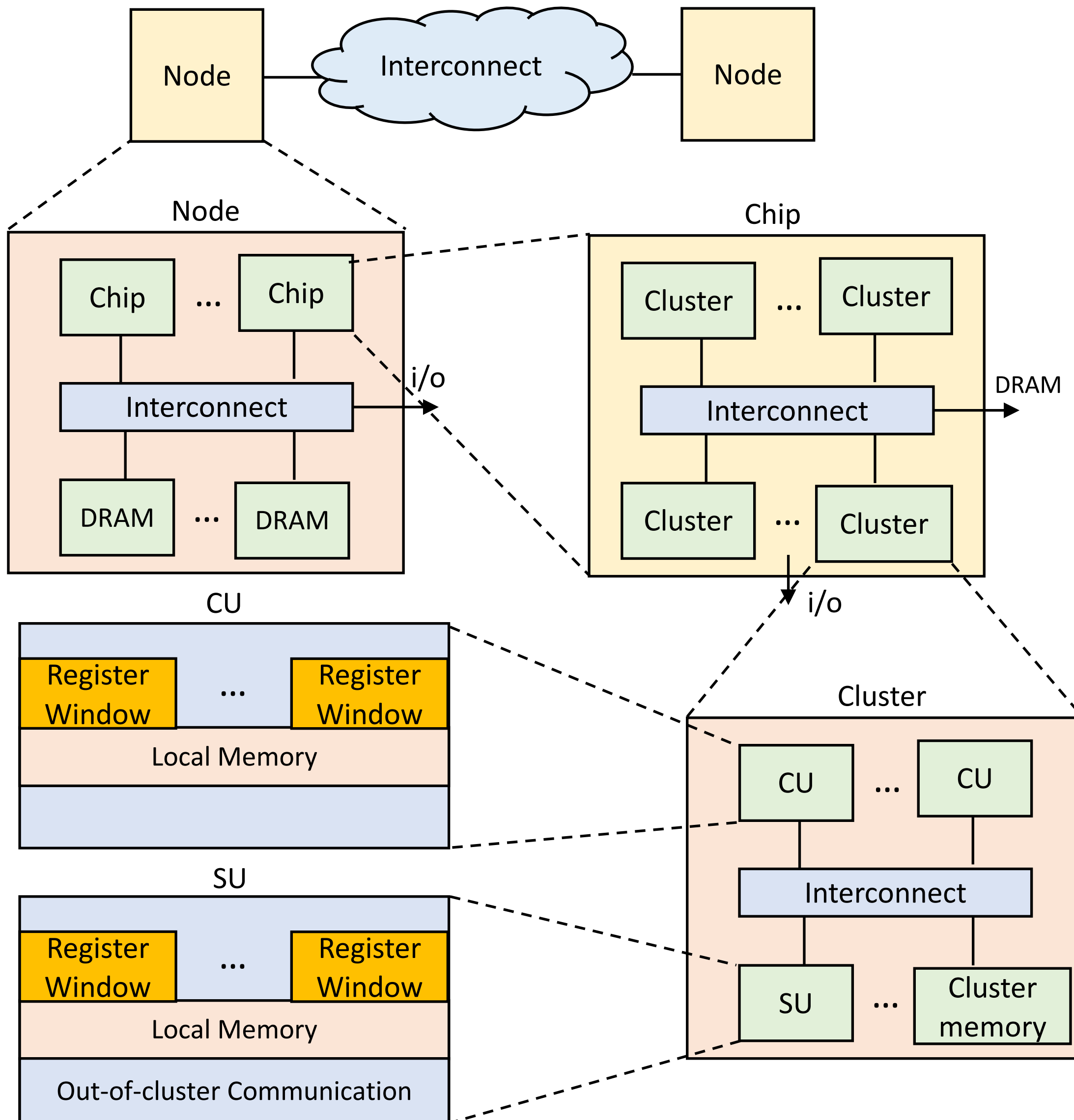


Only Loop Body

(red)	Tokens Iteration 1
(blue)	Tokens Iteration 2
(green)	Tokens Iteration 3

G. R. Gao, “Algorithmic aspects of balancing techniques for pipelined data flow code generation,” *J. Parallel Distrib. Comput.*, vol. 6, pp. 39– 61, Feb. 1989.

Original Codelet Model
Abstract Machine



Firing Rules

- A codelet becomes enabled once tokens (events) are present on each of its input arcs.
- An enabled actor can be fired if it has acquired all its required resource and is scheduled for execution.
- A codelet actor fires by consuming tokens on its input arcs, performing the operations within the codelet, and producing a token (event) on each of its output arcs.

S. Zuckerman, J. Suetterlein, R. Knauerhase, and G. R. Gao, "Using a "codelet" program execution model for exascale machines: Position paper," in *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*, EXADAPT '11, (New York, NY, USA), pp. 64–69, ACM, 2011.

Motivation

Software Pipelining is one of the most successful technology in the exploitation of *Instruction Level Parallelism*

How should the success of software pipelining be exploited under the new many core architecture area?

Unsolved Challenges

Nested Loops

- Numerous techniques exist for single loops (innermost loop) but only few address software pipelining of loop nests.

Multiple Cores

- Traditional software pipelining techniques relied on good static hardware & scheduling techniques which satisfied uniprocessor architectures.
- Challenges to extend this for multiple cores
  - Variability of instruction timing between cores
  - Loop carried dependencies must be realized across different cores.
  - Variable runtime traffic in the on-chip network.

Outline

Background & Motivation

- Software Pipelining
- Dataflow Software Pipelining
- Codelet Model

Solution Methodology

- Extension of Codelet PXM
- Extension of CAM
- Optimizations

Problem Formulation

- Defining Class of Codelet Graph
- Challenges

Future Work

- Implementation of PXM & CAM
- Kernels & Applications

Problem Formulation

Defining Class of Codelet Graphs

A programming model which will leverage
coarse grain parallelism at Codelet graph level &
fine grain parallelism at Codelet level.

At Codelet Graph Level

Class of Codelet graphs for which software pipelining advantages can be clearly demonstrated.

- No Loops.
- Self Loops are allowed.

At Codelet Level

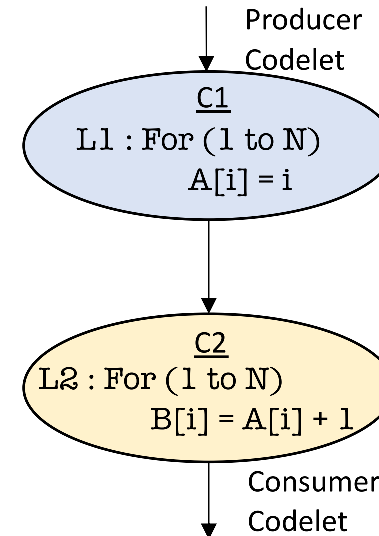
Leverage upon all the work done on the instruction level parallelism & dataflow software pipelining.

- Dependency analyzable loops
- Dependency restricted to affine functions.

Motivating Example
Producer – Consumer Problem

```
#include<iostream>
using namespace std;

int main()

{
    int A[10], B[10];
    int i;

    for(i=0;i<10;i++)
        A[i]=i;

    for(i=0;i<10;i++)
        B[i]=A[i]+1;

    return 1;
}
```

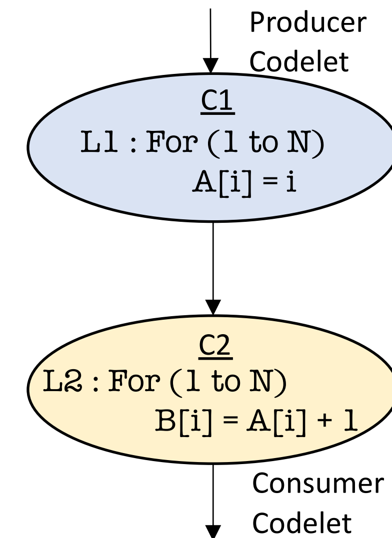
Memory Copy Example



Producer – Consumer Codelets

Motivating Example
Original Codelet Model

- Codelet 1 :
  - Loop with N iterations
- Codelet 2 :
  - loop with N iteration
- Execution :
  - Codelet 1 finishes its execution, sends signal to Codelet 2.
- Output :
  - It will take time required for N + N iterations.
  - We also need to store output of Codelet 1 somewhere and then load it for Codelet 2.

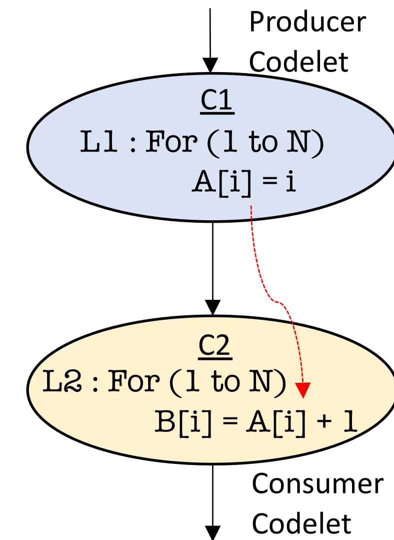


Producer-Consumer Codelets

Motivating Example
Extended Codelet Model with Dataflow Software Pipelining

- Codelet 1 :
  - Loop with N iterations
- Codelet 2 :
  - loop with N iteration
- Execution :
  - With Dataflow Software Pipelining,
    we can start execution of Codelet 2 while Codelet 1 execution is still not finished.

- Output :
  - We can start iteration 1 of Codelet 2 after iteration 1 of Codelet 1 finishes.
  - The first output will be produced at unit time 3.
  - Output will be produced at each iteration after that.



Producer-Consumer Codelets

Challenges
Dataflow Software Pipelining for Codelet Model

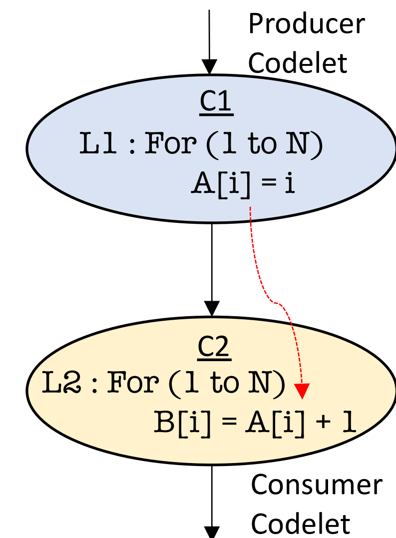
Challenge 1

- Dataflow Software Pipelining behavior described above is ***not*** possible with Original Codelet Model.
  - ***Firing rules*** will not allow Consumer Codelet to fire before Producer Codelet finishes its execution.

- Such a behavior will be very useful for ***Streaming*** applications.

- Solution Methodology
  - Extension to kinds of *events*
  - Extension to *firing rules*.

Extended Codelet Program Execution Model



Producer-Consumer Codelets

Challenges

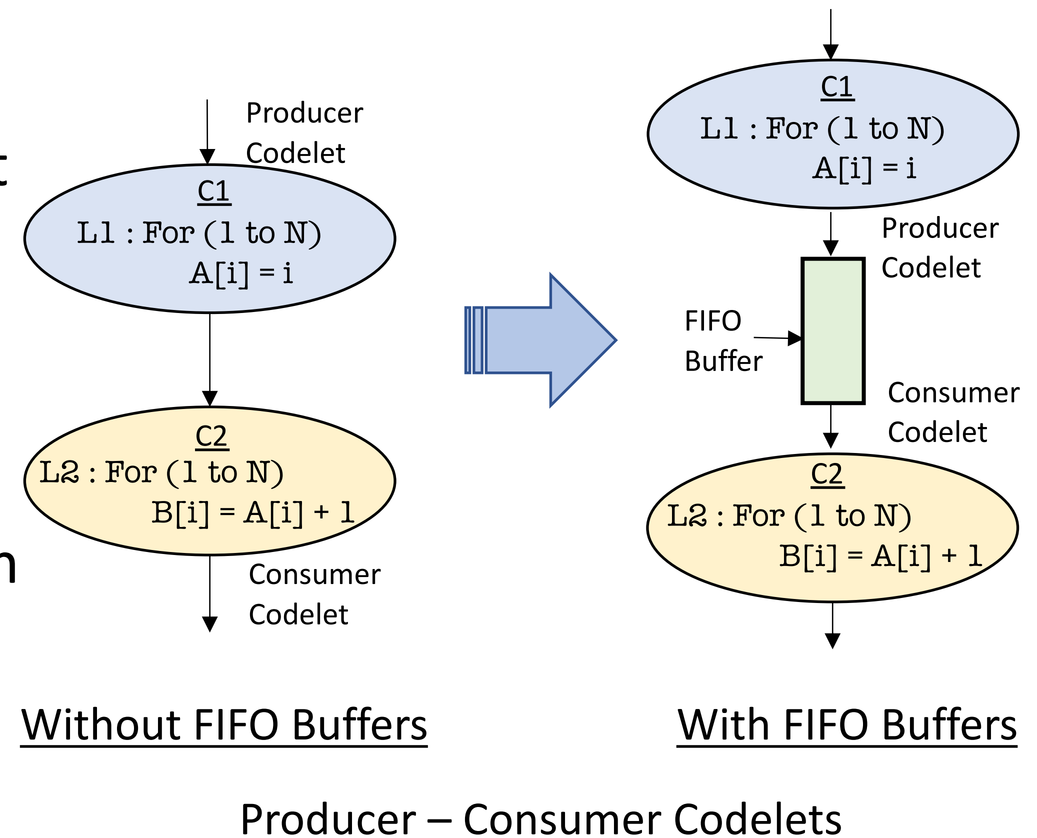
Dataflow Software Pipelining for Codelet Model

Challenge 2

- Software implementation of these extension to support Dataflow Software Pipelining in Codelet Program Execution Model will incur some additional *cost*.

- ***Hardware-Software Co-Design*** to efficiently support extension to Codelet Program Execution Model.

Extended Codelet Abstract Machine Model



Producer – Consumer Codelets

Outline

Background & Motivation

- Software Pipelining
- Dataflow Software Pipelining
- Codelet Model

Solution Methodology

- Extension of Codelet PXM
- Extension of CAM
- Optimizations

Problem Formulation

- Defining Class of Codelet Graph
- Challenges

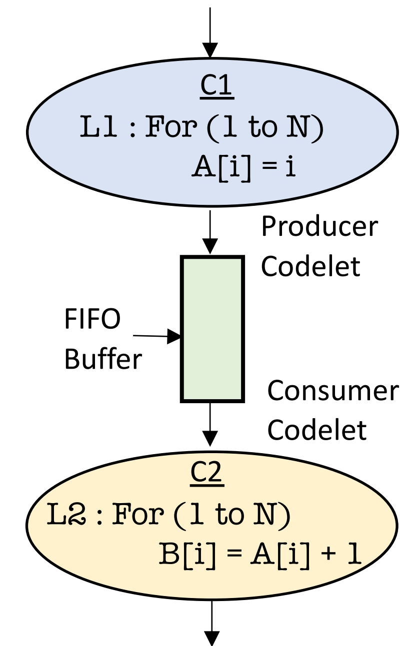
Future Work

- Implementation of PXM & CAM
- Kernels & Applications

Solution Methodology
Step 1: Extending Codelet Program Execution Model.

Extending Firing Rules

- Under certain cases, We will allow next Codelet to begin its execution before the current Codelet has finished its execution.
  - *Producer-consumer* behavior
  - *forall* loop
- Codelet level API to support FIFO operations.
- Decided by compiler or explicitly indicated by programmer.

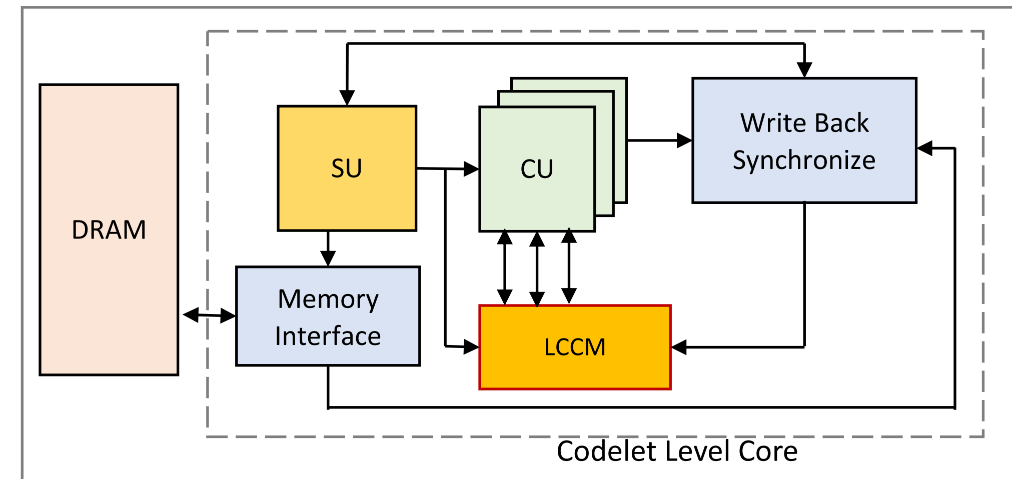


Producer-Consumer Codelets

Solution Methodology
Step 2 : Extending Codelet Abstract Machine

- Ongoing work on *Sequential Codelet Model (SCM) for parallel code execution.* [Monsalve et. al 2019]

- Focus on ***Codelet Level Core***
  - FIFO buffers can be envisioned in the *Local Codelet Level Core Memory* (LCCM)
  - Dataflow Software Pipelining at Threaded Procedure (TP) level.

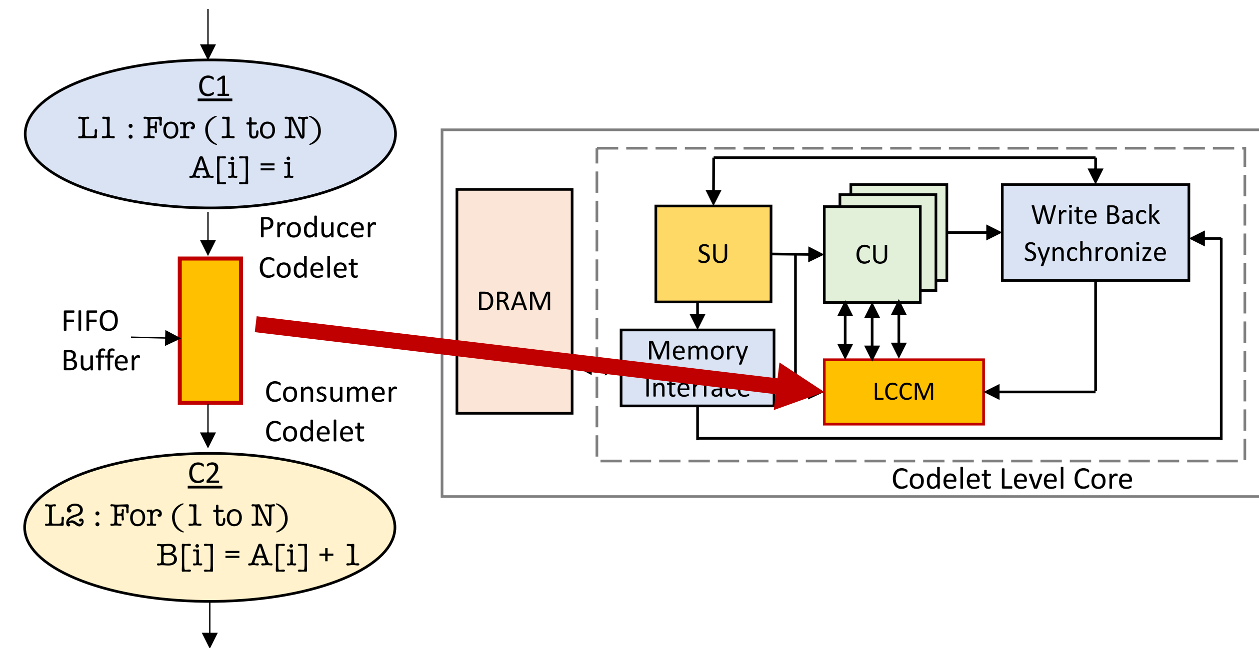


[Monsalve et. al 201] "Sequential Codelet Model for Parallel Execution" , Jose M Monsalve Diaz, and Guang R Gao , CAPSL Technical Memo 132, June 2019.

Solution Methodology
Step 2 : Extending Codelet Abstract Machine

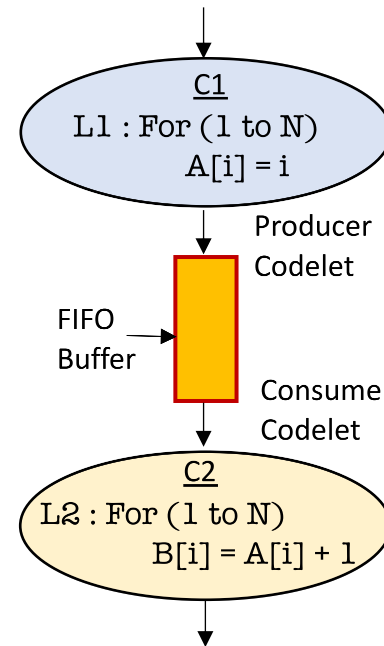
- Ongoing work on *Sequential Codelet Model (SCM) for parallel code execution.* [Monsalve et. al 2019]

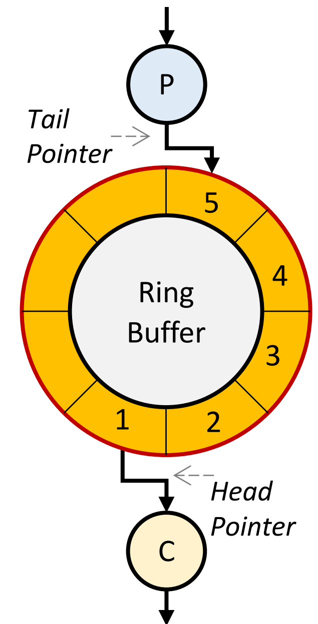
- Focus on ***Codelet Level Core***
  - FIFO buffers can be envisioned in the *Local Codelet Level Core Memory* (LCCM)
  - Dataflow Software Pipelining at Threaded Procedure (TP) level.



Solution Methodology
Step 3: Optimizations

- A *circular buffer* or a *ring buffer* is efficient data structure for *implementation* of FIFO buffer.
- elements are NOT shifted when one is consumed.
- Only *head* & *tail* pointers are updated every time data is added/ removed.





FIFO Ring Buffers

All Codelet graphs are not simplistic like Single producer – Single Consumer

Single Producer – Multiple Consumer

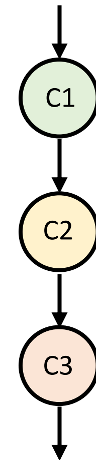
- ***Chain of Codelets.***
  - C1 is the only *producer* codelet.
  - C2 producer as well as consumer Codelet.
  - C3 is consumer Codelet.
- ***Tree of Codelets.***
  - P is the only *producer* codelet.
  - C1, C2 & C3 are consumer codelets.
  - They consume the same data tokens produced by codelet P.

Do you duplicate tokens & allocated separate FIFO buffer for each arc?

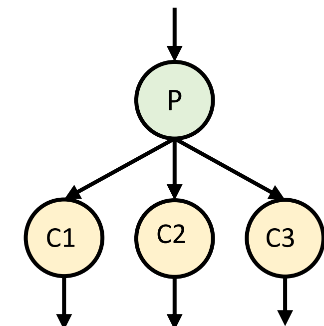
Or

You can find mechanism to share FIFO buffers?

Chain of Codelets

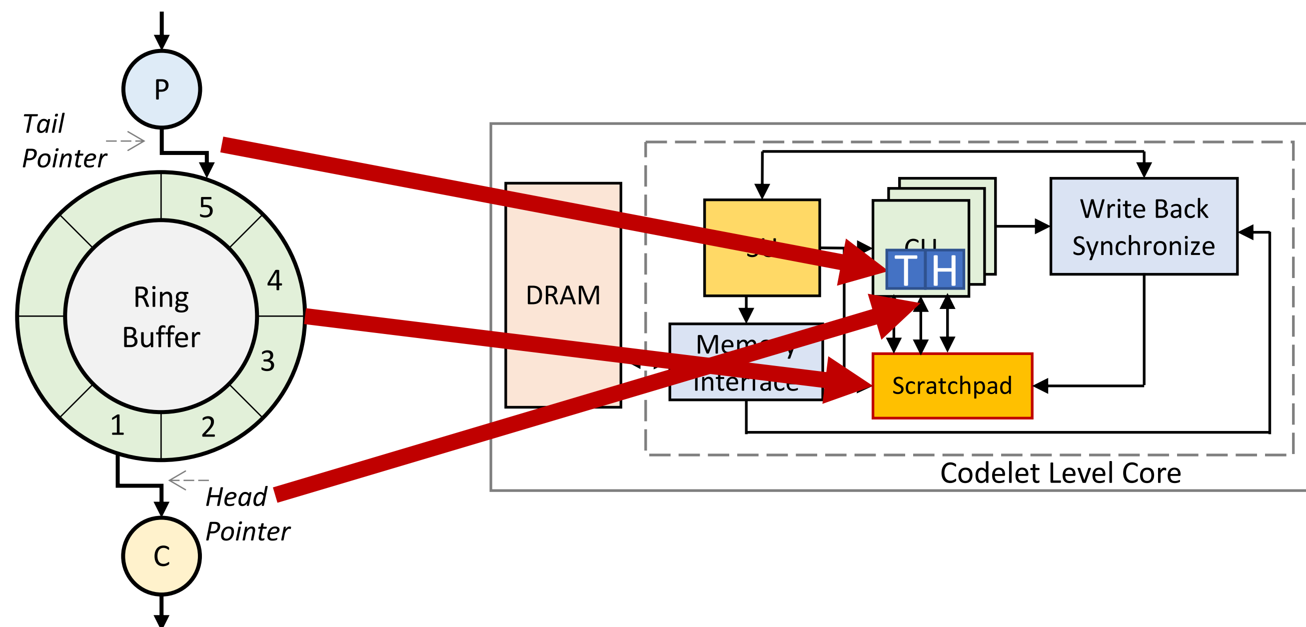


Tree of Codelets



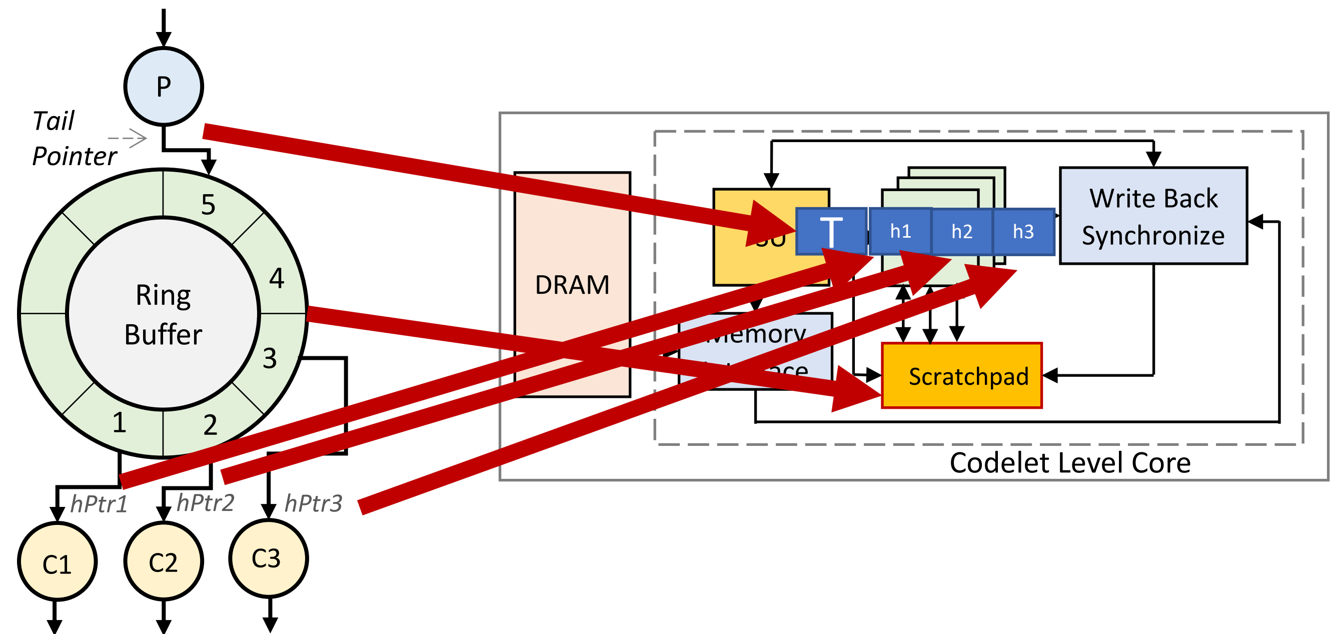
FIFO Ring Buffer

- Ring Buffer will be allocated in the LCCM (Scratchpad)
- The *tail* and *head* pointers will be maintained in the specific registers



Multiple-Head FIFO Ring Buffers

- Share FIFO Ring Buffer with all consumers of same producer
- Each Codelet will maintain head pointers in registers in their respective CUs.
- There will be only 1 tail pointer for 1 producer.



Outline

Background & Motivation

- Software Pipelining
- Dataflow Software Pipelining
- Codelet Model

Solution Methodology

- Extension of Codelet PXM
- Extension of CAM
- Optimizations

Problem Formulation

- Defining Class of Codelet Graph
- Challenges

Future Work

- Implementation of PXM & CAM
- Kernels & Applications

Future Work

Implementation : Software

- Codelet Level Instruction Set Architecture (ISA) to leverage FIFO buffers.
- Extension to DARTS to support Dataflow Software Pipelining.

Implementation : Hardware-Software

- FPGA Implementation
  - DEMAC cluster [Roa et al. 2018]

Kernels & Applications

- Cannons algorithm for Matrix Multiplication.

[Roa et al. 2018] Diego Roa, Jose Monsalve, Ryan Kabrick, Xu Tan, and Guang Gao (2018). *Landing Codelets PXM on DARTS, DEMAC - Towards a Hardware/Software, Codesign/Evaluation Platform*. University of Delaware booth, Supercomputing 2018, November 2018, Dallas, Texas.

THANK YOU.

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.