Communication between Nested Loop Programs via Circular Buffers in an Embedded Multiprocessor System

Tjerk Bijlsma[2], Marco Bekooij[1], Pierre Jansen[2], Gerard Smit[2]

1

founded by Philips

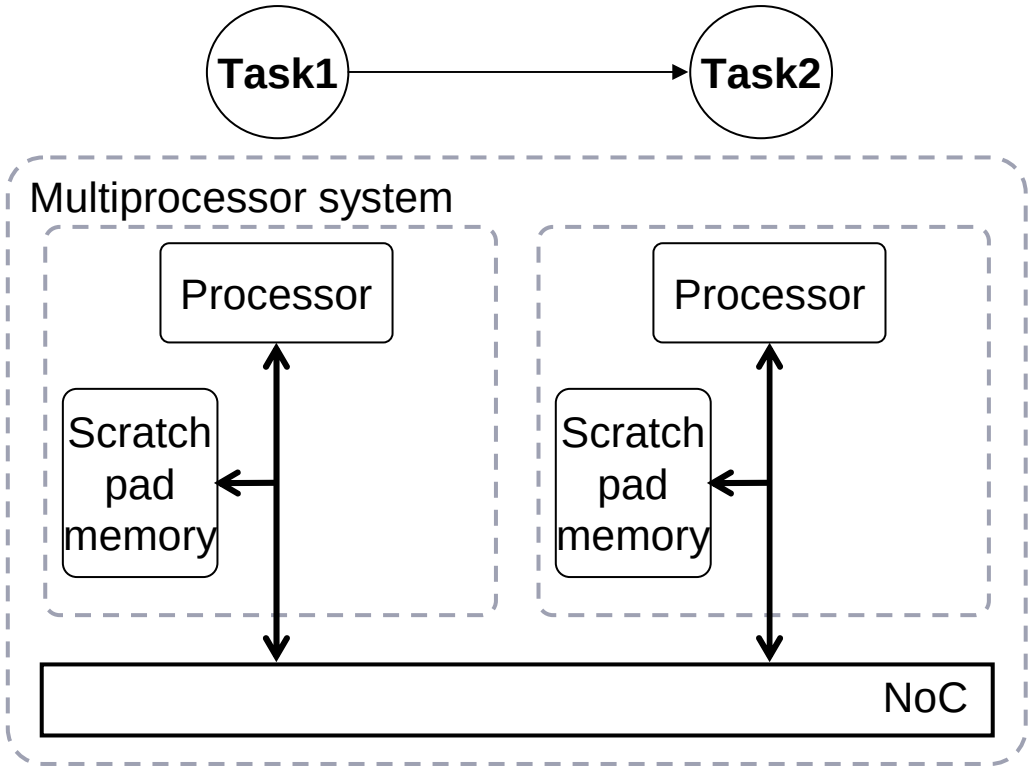
2

University of Twente
Enschede - The Netherlands

Outline

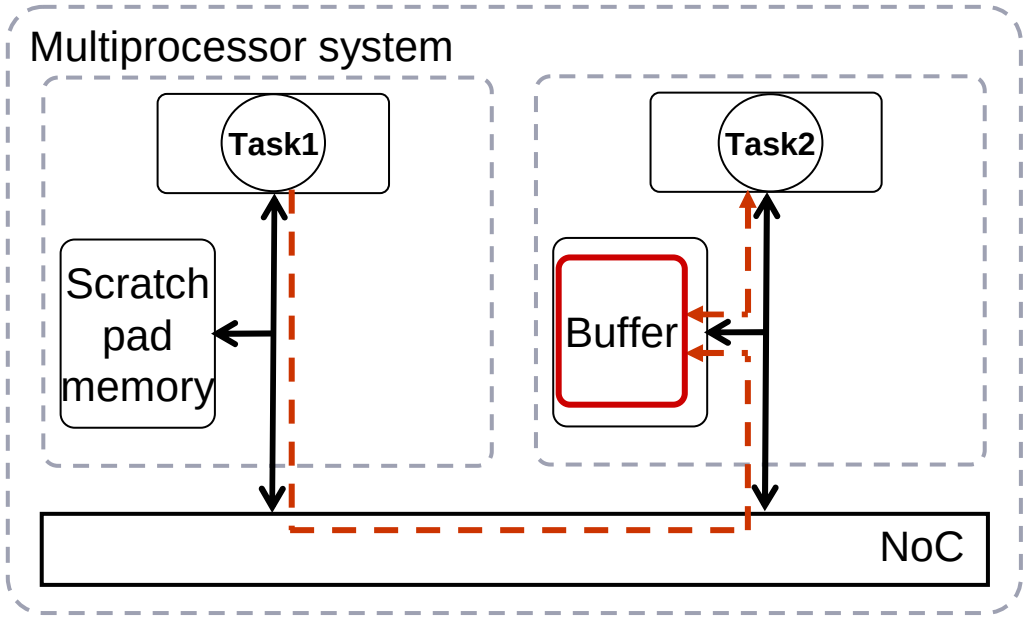
- Problem statement
- Target application
- Related methods
- Communication via circular buffers
  - Circular buffer
  - Sliding windows
  - Extending the nested loop programs
  - Buffer capacities
- Case study
- Conclusion

Problem statement



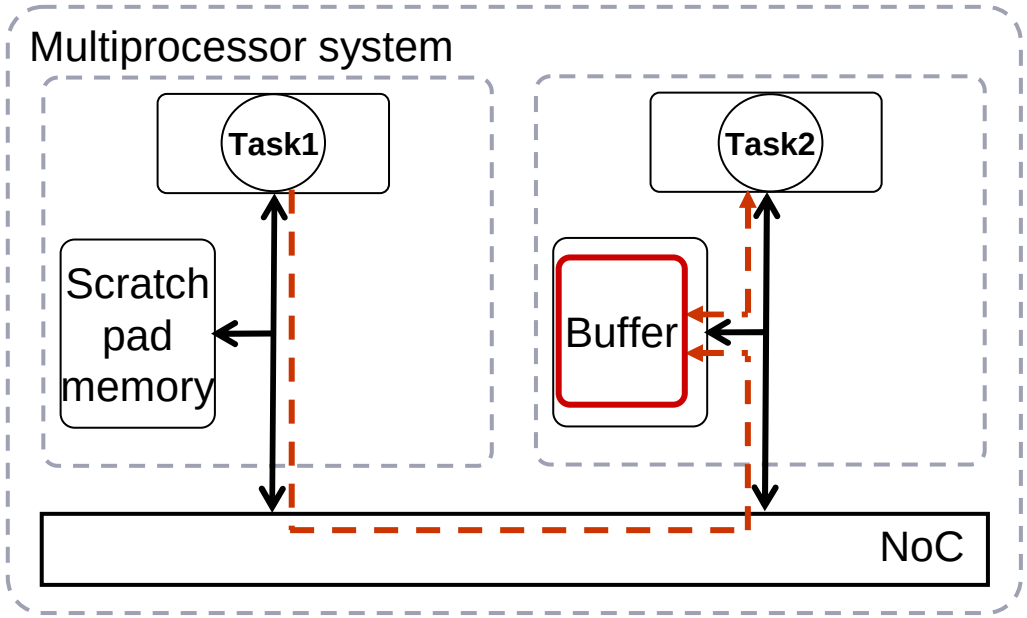
- How to arrange communication in a multiprocessor system?
  - Application is a task graph
  - A task contains a nested loop program
  - Nested loop programs communicate by writing and reading arrays

Solution direction



- Organize the communication and synchronization of arrays via buffers
- Locate the buffer in the scratch pad memory of the consuming task

Remaining issues



- How do the tasks use the buffer?
- How to determine the buffer capacity?

Target Application

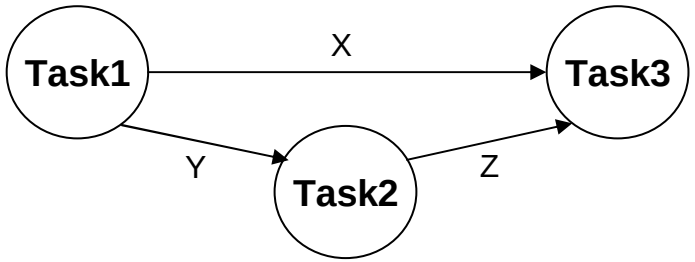
Target Application

- Application is an acyclic task graph
- Task contains a nested loop program
  - Single assignment code
  - Side effect free functions
  - Constant loop bounds
  - Index expressions with iterators as variables
    - ***Not*** *limited to affine index expressions*

Application

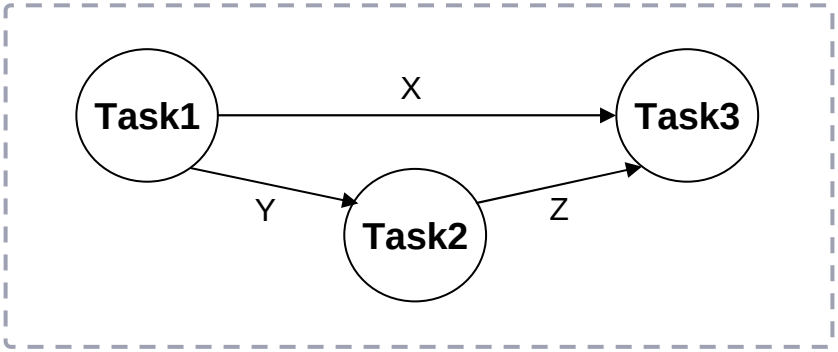
```
int X[12];
int Y[12];
for i0:0:5
 for i1:0:1{
  X[2i0-i1+1] = F1(~);
  Y[2i0-i1+1] = F2(~);
 }
```

```
int X[12];
int Z[14];
for k0:0:3
 for k1:0:2{
  ~ = F3(X[11-3k0-k1]);
  ~ = F4(Z[3k0+k1]);
 }
```




```
int Y[12];
int Z[12];
for j0:0:1
 for j1:0:6
  Z[7j0+j1] = F5(Y[5j0+j1]);
```

Target Application



- Communication granularity
  - Locations
- Access patterns in array
  - **Out-of-order access**

Production into Y (α(task1,Y,i))

0	1	2	3	4	5	6	7	8	9	10	11
1	0	3	2	5	4	7	6	9	8	11	10

  - **Multiplicity**

Consumption from Y (α(task2,Y,i))

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	2	3	4	**5**	**6**	**5**	**6**	7	8	9	10	11

  - **Skipping**

Consumption from Z (α(task3,Z,i))

0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11

Production into Z (α(task2,Z,i))

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Related methods

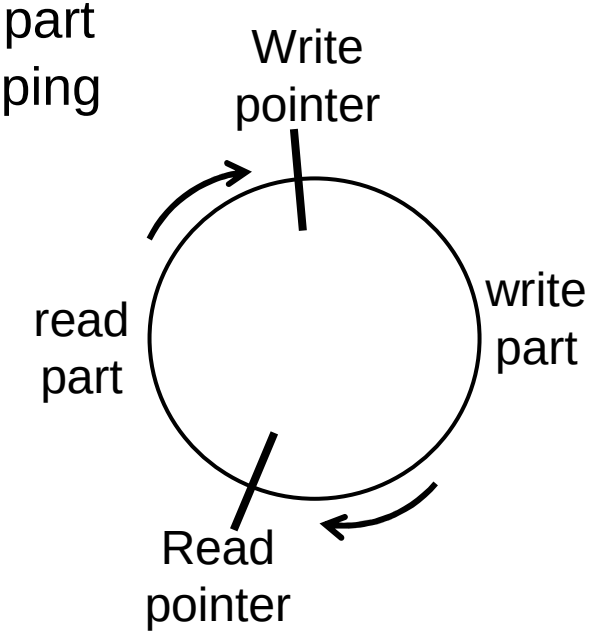
Related methods

- Single window (E. de Greef et.al., 1997 and D. Gannon et.al. 1988) :
  - Window contains values after they are written until they are read
  - Single processor solution
  - Requires affine index expressions
- FIFO buffer (A. Turjan et.al., 2004) :
  - Depending on access pattern a reordering memory and process are required
  - Requires affine index expressions
- Read and write window (K. Huang et.al., 2007) :
  - Provides no analysis to determine either window sizes or deadlock freedom of application

Communication via circular buffers

Communication via circular buffers

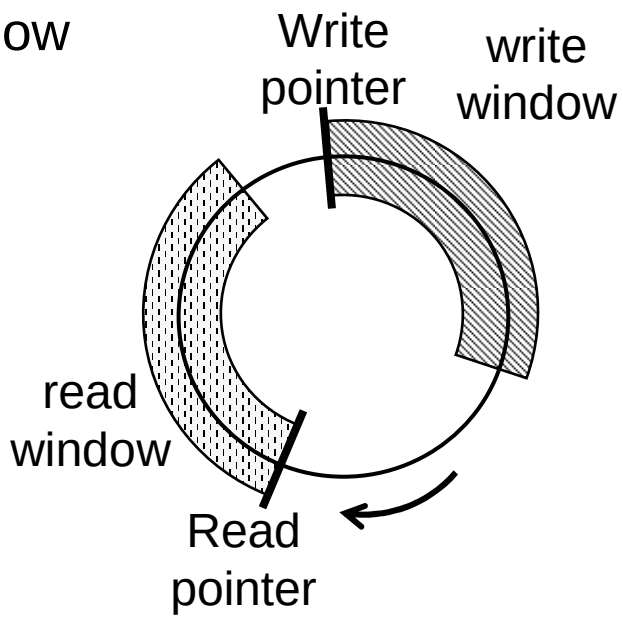
- Circular buffer
  - Contains a read and a write pointer
  - Mutual exclusive access in either, read or write part
  - Allows out-of-order access, multiplicity and skipping



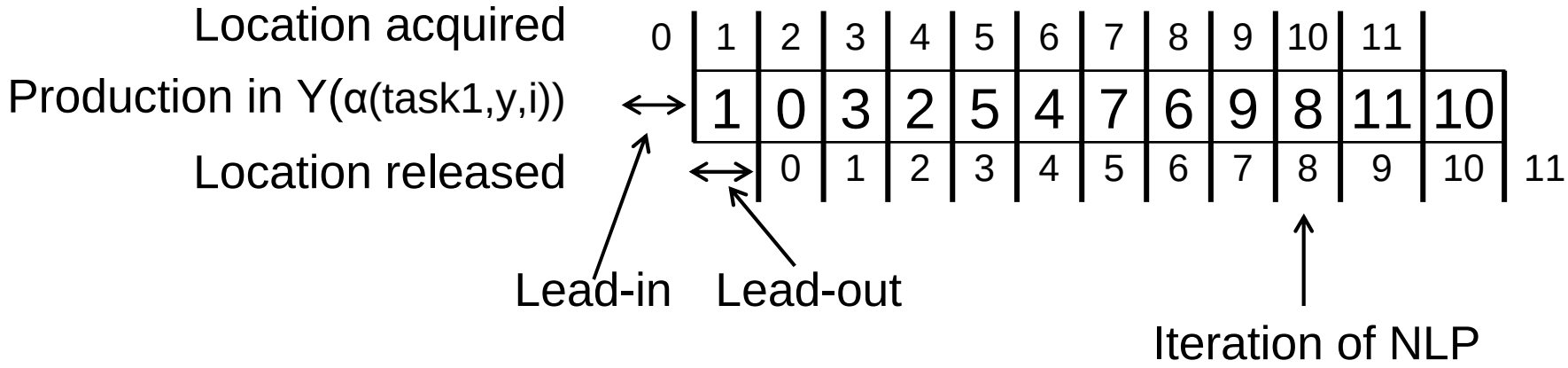
- Synchronization
  - Requires a memory consistency model
    (Streaming memory consistency [Brand et. al.,DSD 2007])
    - Acquire and release each location in the buffer
    - Allows posted writes

Communication via circular buffers

- Circular buffer + synchronization
  - A sequence of acquired locations
  - Every iteration of a nested loop program conditionally acquires and releases one location
  - A nested loop program acquires and releases consecutive locations
    - Results in a sliding read and write window



Communication via circular buffers



- Content of a sliding window
  - Lead-in ($d_1(t,b)=\max_i\{\alpha(t,b,i)-i\}$)
    - Number of initially acquired locations
    - Guarantees that accessed location is acquired
  - Lead-out ($d_2(t,b)=\max_i\{i-\alpha(t,b,i)\}$)
    - Number of initial iterations without a release
    - Guarantees that a location is not released before the last access
- Window size : $d_1 + d_2 + 1$

Communication via circular buffers

- Extending the nested loop programs to use sliding windows
  - Add acquire and release statements
  - Replace read and write statements

```
int X[12];
int Y[12];
for i0:0:5
 for i1:0:1{
   X[2i0-i1+1] = F1(~);
   Y[2i0-i1+1] = F2(~);
 }
```

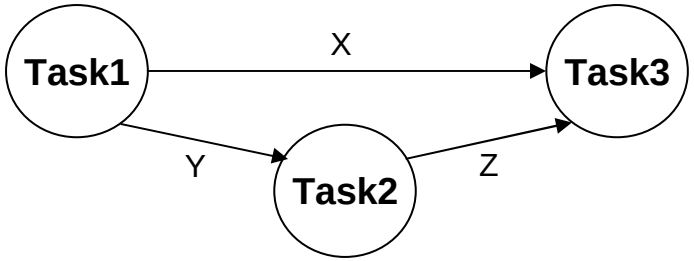
Buffer	X	Y
Lead-in (d_1)	1	1
Lead-out (d_2)	1	1

```
int c1 = 1;
acquire(1,X);
acquire(1,Y);
for i0:0:5
 for i1:0:1 {
  if(c1 < 12)
   acquire(1,X);
  if(c1 < 12)
   acquire(1,Y);
  write(2i0-i1+1,X,F1(~));
  write(2i0-i1+1,Y,F2(~));
  if(c1 > 1 )
   release(1,X);
  if(c1 > 1)
   release(1,Y);
  c1++;
 }
release(1,X);
release(1,Y);
```

Communication via circular buffers

- Global notion of synchronization to determine the buffer capacities
  - Acquire operation in buffer Z depends upon acquire in buffer X

```
int c1 = 1;
acquire(1,X);
acquire(1,Y);
for i0:0:5
 for i1:0:1 {
  if(c1 < 12)
   acquire(1,X);
  if(c1 < 12)
   acquire(1,Y);
  write(2i0-i1+1,X,F1(~));
  write(2i0-i1+1,Y ,F2(~));
  if(c1 > 1 )
   release(1,X);
  if(c1 > 1)
   release(1,Y);
  c1++;
 }
release(1,X);
release(1,Y);
```

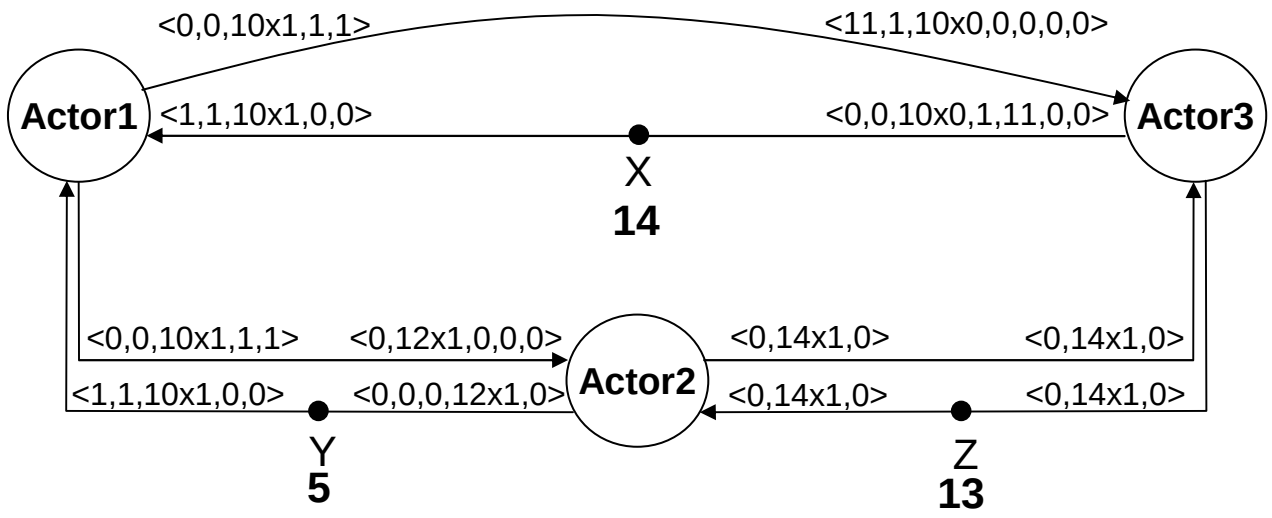



```
int c2 = 1
acquire(0,Y);
acquire(0,Z);
for j0:0:1
 for j1:0:6{
  if (c2 < 13)
   acquire(1,Y);
  if (c2 < 15)
   acquire(1,Z);
  write(7j0+j1,Z,
  F5(read(5j0+j1,Y)));
  if (c2 < 2)
   release(1,Y);
  if (c2 < 0)
   release(1,Z);
  c2++;
 }
release(1,Y);
release(0,Z);
```

```
int c3 = 1
acquire(11,X);
acquire(0,Z);
for k0:0:3
 for k1:0:2{
  if(c3 < 2)
   acquire(1,X);
  if(c3 < 13)
   acquire(1,Z)
  ~ = F3(read(11-3k0-k1,X));
  ~ = F4(read(3k0+k1,Z));
  if(c3 > 11)
   release(1,X);
  if(c3 > 0)
   release(1,Z);
  c3++
 }
release(11,X);
for k3:0:1:1 {
 acquire(1,Z);
 release(1,Z);
}
```

Communication via circular buffers

- Cyclo static dataflow model gives global notion of synchronization
  - Execution of a loop-nest shows periodic behavior
  - Sufficient buffer capacities can be calculated [Wiggers et.al.,DAC2007]



Array	Size
X	12
Y	12
Z	14

Case study

Case study

- A fragment of a digital audio broadcasting (DAB) channel decoder
  - Demapper (DM)
  - Frequency deinterleaver (FDI)
    - Pseudo random function $F(j_0)$ determines read location
  - Time deinterleaver (TDI)


```
int x[2048];
for i0:0:2047{
   x[i0] = ~;
}
```

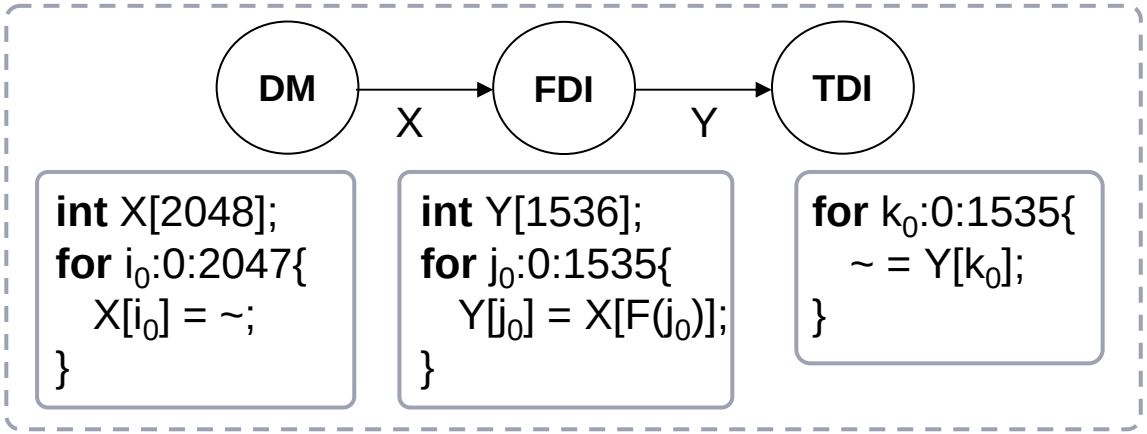
```
int y[1536];
for j0:0:1535{
   y[j0] = x[F(j0)];
}
```

```
for k0:0:1535{
   ~ = y[k0];
}
```

```
int F(int index){
  int c=0,val=511;
  while (c < index){
    val = ((val * 13)+511)%2048;
    if ((val ≥ 256)/\
     (value ≠ 1024)/\(val ≤ 1792))
      c++;
  }
  return val;
}
```

Case study

- Sufficient buffer capacities to guarantee deadlock free execution
  - Capacity of buffer:
    - X = 3464
    - Y = 388



Task	DM	FDI		TDI
Buffer	X	X	Y	Y
Lead-in (d_1)	0	1713	0	0
Lead-out (d_2)	0	1235	0	0

Conclusions

- We used sliding windows in a circular buffer for inter-task communication
  - Suitable for nested loop programs with non-affine index expressions
  - Use a read and a write window
    - Allows out-of-order access, multiplicity and skipping patterns

- Sufficient buffer capacities computed with the derived cyclo static dataflow model
  - Guarantee deadlock free execution

- Demonstrated approach on a fragment of a digital audio broadcasting (DAB) channel decoder
  - We could handle the pseudo random read function of frequency deinterleaver

?