Sheep in wolf's clothing: Implementation models for dataflow multi-threaded software

Keryan Didier [1] Albert Cohen [2] Adrien Gauffriau [3]
Dumitru Potop-Butucaru [1]

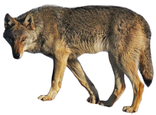
[1]INRIA [2]ENS & Google [3]Airbus

ACSD'19 27 June 2019

Critical systems context

Threaded implementation

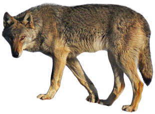
Posix threads	
Non-determinism	–
Data races possible	–
Deadlocks possible	–
Asynchronous small-step semantics	–
Portability Flexibility General-purpose	+++



Critical systems context

Threaded implementation and dataflow synchronous specification

Posix threads			Dataflow synchronous formalisms
Non-determinism	−	+	Determinism
Data races possible	−	+	No data races
Deadlocks possible	−	+	No deadlocks
Asynchronous small-step semantics	−	+	Big-step semantics
Portability Flexibility General-purpose	+++		



Critical systems context

Threaded implementation and dataflow synchronous specification

Posix threads			Dataflow synchronous formalisms
Non-determinism	−	+	Determinism
Data races possible	−	+	No data races
Deadlocks possible	−	+	No deadlocks
Asynchronous small-step semantics	−	+	Big-step semantics
Portability Flexibility General-purpose	+++		


Our thesis : in practice, threaded implementations of dataflow specification preserve a fundamentally dataflow structure

Application

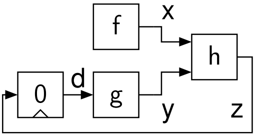
Dataflow synchronous specification in Lustre

```
fun f:()->(float)
fun g:(int)->(int)
fun h:(float,int)->(int)
var
  x : float; y : int; z : int; d : int;
let
  x = f();
  y = g(d);
  z = h(x,y);
  d = 0 fby z;
tel
```

Application

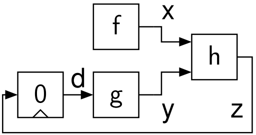
Dataflow synchronous specification in Lustre

```
fun f:()->(float)
fun g:(int)->(int)
fun h:(float,int)->(int)
var
  x : float; y : int; z : int; d : int;
let
  x = f();
  y = g(d);
  z = h(x,y);
  d = 0 fby z;
tel
```

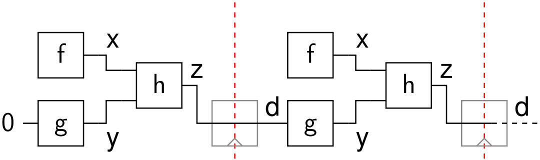


Application

Dataflow synchronous specification in Lustre

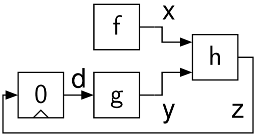



```
fun f:()->(float)
fun g:(int)->(int)
fun h:(float,int)->(int)
var
  x : float; y : int; z : int; d : int;
let
  x = f();
  y = g(d);
  z = h(x,y);
  d = 0 fby z;
tel
```



Application

Two cores implementation (static allocation and scheduling)

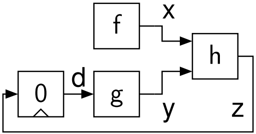



```
void thread_cpu0(){
  for(;;){
    f(&x);
    dcache_flush();



    mutex_lock(&m0);
    dcache_inval();
    h(x,y,&z);
    dcache_flush();
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();
    g(z,&y);
    dcache_flush();
    mutex_unlock(&m0);






}}
```

Application

Two cores implementation (static allocation and scheduling)




```
void init(){
  z = 0;
  mutex_unlock(&m1);
}
```

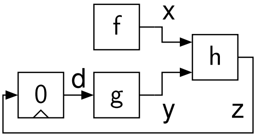
```
void thread_cpu0(){
  for(;;){
    f(&x);
    dcache_flush();


    mutex_lock(&m0);
    dcache_inval();
    h(x,y,&z);
    dcache_flush();
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();
    g(z,&y);
    dcache_flush();
    mutex_unlock(&m0);



}}
```

Application

Two cores implementation (static allocation and scheduling)

```
ldscript fragment:
  x=0x22000; y=0x32000; z=0x22004; i=0x220064;
  stack0=0x30000; stack1=0x40000;
  .=0x20000; .bank2:{*(.text.cpu0);
                   .=0x100 ; *(.text.f) ;
                   .=0x500 ; *(.text.h) ;
  }
  .=0x30000; .bank3:{*(.text.cpu1);
                   .=0x200 ; *(.text.g) ;
  }
```




```
void init(){
  z = 0;
  mutex_unlock(&m1);
}
```

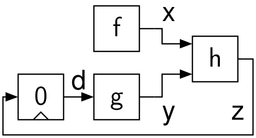
```
void thread_cpu0(){
  for(;;){
    f(&x);
    dcache_flush();



    mutex_lock(&m0);
    dcache_inval();
    h(x,y,&z);
    dcache_flush();
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();
    g(z,&y);
    dcache_flush();
    mutex_unlock(&m0);






}}
```

Application

Two cores implementation (static allocation and scheduling)

```
ldscript fragment:
  x=0x22000; y=0x32000; z=0x22004; i=0x220064;
  stack0=0x30000; stack1=0x40000;
  .=0x20000; .bank2:{*(.text.cpu0);
                   .=0x100 ; *(.text.f) ;
                   .=0x500 ; *(.text.h) ;
  }
  .=0x30000; .bank3:{*(.text.cpu1);
                   .=0x200 ; *(.text.g) ;
  }
```




```
void init(){
  z = 0;
  mutex_unlock(&m1);
}
```

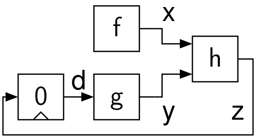
```
void thread_cpu0(){
  for(;;){
    f(&x);
    dcache_flush();


    mutex_lock(&m0);
    dcache_inval();
    h(x,y,&z);
    dcache_flush();
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();
    g(z,&y);
    dcache_flush();
    mutex_unlock(&m0);
}}
```

Application

Two cores implementation (static allocation and scheduling)

```
ldscript fragment:
  x=0x22000; y=0x32000; z=0x22004; i=0x220064;
  stack0=0x30000; stack1=0x40000;
  .=0x20000; .bank2:{*(.text.cpu0);
                      .=0x100 ; *(.text.f) ;
                      .=0x500 ; *(.text.h) ;
  }
  .=0x30000; .bank3:{*(.text.cpu1);
                      .=0x200 ; *(.text.g) ;
  }
```




```
void init(){
  z = 0;
  mutex_unlock(&m1);
}
```

```
void thread_cpu0(){
  for(;;){
    f(&x);
    dcache_flush();

    mutex_lock(&m0);
    dcache_inval();
    h(x,y,&z);
    dcache_flush();
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();
    g(z,&y);
    dcache_flush();
    mutex_unlock(&m0);
}}
```

Dataflow representation of the implementation

Functional aspects

```
void init(){
  z = 0;                                   d = z fby 0
  mutex_unlock(&m1);
}
```

```
void thread_cpu0(){
  for(;;){
    f(&x);                         x  = f()
    dcache_flush();

    mutex_lock(&m0);
    dcache_inval();
    h(x,y,&z);                     z  = h(x ,y )
    dcache_flush();
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();
    g(z,&y);                       y  = g(d )
    dcache_flush();
    mutex_unlock(&m0);
}}
```

- Specification dataflow

Application

Two cores implementation (static allocation and scheduling)

```
void init(){
  z = 0;                                   d = z fby 0
  mutex_unlock(&m1);
}
```

```
void thread_cpu0(){
  for(;;){
    f(&x);                        x0 = f()
    dcache_flush();                x = x0

    mutex_lock(&m0);
    dcache_inval();               y0 = y
    h(x,y,&z);                    z0 = h(x0,y0)
    dcache_flush();                z = z0
    mutex_unlock(&m1);
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);
    dcache_inval();                          d1 = d
    g(z,&y);                                 y1 = g(d1)
    dcache_flush();                           y = y1
    mutex_unlock(&m0);
}}
```

- Memory hierarchy representation (variable duplications)

Application

Two cores implementation (static allocation and scheduling)

```
void init(){
  z = 0;                                  d = z fby 0
  mutex_unlock(&m1);                     u1 = top fby u
}
```

```
void thread_cpu0(){
  for(;;){
    f(&x);                               x0 = f()
    dcache_flush();                       x = x0

    mutex_lock(&m0);                      _ = v
    dcache_inval();                      y0 = y
    h(x,y,&z);                           z0 = h(x0,y0)
    dcache_flush();                       z = z0
    mutex_unlock(&m1);                    u = top
}}
```

```
void thread_cpu1(){
  for(;;){
    mutex_lock(&m1);                      _ = u1
    dcache_inval();                      d1 = d
    g(z,&y);                             y1 = g(d1)
    dcache_flush();                       y = y1
    mutex_unlock(&m0);                    v = top
}}
```

- Mutex operations

Dataflow representation of the implementation

Dataflow synchronous implementation model

```
fun f:()->(float)
fun g:(int)->(int)
fun h:(float,int)->(int)
var
  x:int y:float z:int d:int
  x0:int y0:float y1:float
  z0:int d1:int
  u:event u1:event v:event
let
  d = 0 fby z
  u1 = top fby u

  x0 = f()
   x = x0

   _ = u1
  d1 = d
  y1 = g(d1)
   y = y1
   v = top

   _ = v
  y0 = y
  z0 = h(x0,y0)
   z = z0
   u = top
tel
```

- Statements all interpreted, but no structure nor sequencing

Dataflow representation of the implementation

Extended dataflow synchronous implementation model

```
fun f:()->(float)
fun g:(int)->(int)
fun h:(float,int)->(int)
var
  x:int y:float z:int d:int
  x0:int y0:float y1:float
  z0:int d1:int
  u:event u1:event v:event s0,s1,s2,s3,s4,s5,s6,s7,t0,t1,t2,t3:event
let
  d = 0 fby z
  u1 = top fby u
  s7 = top fby s6
```

```
    wait(s7) done(s0)                    x0 = f()
    wait(s0) done(s1)                     x = x0
```

```
    done(t0)                              _ = u1
    wait(t0) done(t1)                    d1 = d
    wait(t1) done(t2)                    y1 = g(d1)
    wait(t2) done(t3)                     y = y1
    wait(t3)                              v = top
```

```
    wait(s1) done(s2)                     _ = v
    wait(s2) done(s3)                    y0 = y
    wait(s3) done(s4)                    z0 = h(x0,y0)
    wait(s4) done(s5)                     z = z0
    wait(s5) done(s6)                     u = top
tel
```

- Explicit sequencing
- Functionally complete : same traces as C code under async. semantics

Dataflow representation of the implementation

Non-functional annotations

```
fun f:()->(float)
fun g:(int)->(int)
fun h:(float,int)->(int)
var
  x:int y:float z:int d:int
  x0:int y0:float y1:float
  z0:int d1:int
  u:event u1:event v:event s0,s1,s2,s3,s4,s5,s6,s7,t0,t1,t2,t3:event
let
  d = 0 fby z
  u1 = top fby u
  s7 = top fby s6
thread
    wait(s7) done(s0)          x0 = f()
    wait(s0) done(s1)           x = x0

    wait(s1) done(s2)           _ = v
    wait(s2) done(s3)          y0 = y
    wait(s3) done(s4)          z0 = h(x0,y0)
    wait(s4) done(s5)           z = z0
    wait(s5) done(s6)           u = top
tel
```

```
thread
    done(t0)                    _ = u1
    wait(t0) done(t1)          d1 = d
    wait(t1) done(t2)          y1 = g(d1)
    wait(t2) done(t3)           y = y1
    wait(t3)                    v = top
```

- Thread structure

Dataflow representation of the implementation

Non-functional annotations

```
fun f:()->(float) at 0x20100
fun g:(int)->(int) at 0x30300
fun h:(float,int)->(int) at 0x20500
var
  x:int at 0x22000 y:float at 0x32000 z:int at 0x22004 d:int at z
  x0:int at x on cpu0 y0:float at y on cpu0 y1:float at y on cpu1
  z0:int at z on cpu0 d1:int at z on cpu1
  u:event at m1 u1:event at m1 v:event at m0 s0,s1,s2,s3,s4,s5,s6,s7,t0,t1,t2,t3:event
let
  d = 0 fby z
  u1 = top fby u
  s7 = top fby s6
thread on cpu0 at 0x20000 stack 0x30000
    wait(s7) done(s0)          x0 = f()
    wait(s0) done(s1)           x = x0

    wait(s1) done(s2)           _ = v
    wait(s2) done(s3)          y0 = y
    wait(s3) done(s4)          z0 = h(x0,y0)
    wait(s4) done(s5)           z = z0
    wait(s5) done(s6)           u = top
thread on cpu1 at 0x30000 stack 0x40000
    done(t0)                    _ = u1
    wait(t0) done(t1)          d1 = d
    wait(t1) done(t2)          y1 = g(d1)
    wait(t2) done(t3)           y = y1
    wait(t3)                    v = top
tel
```

- Allocation

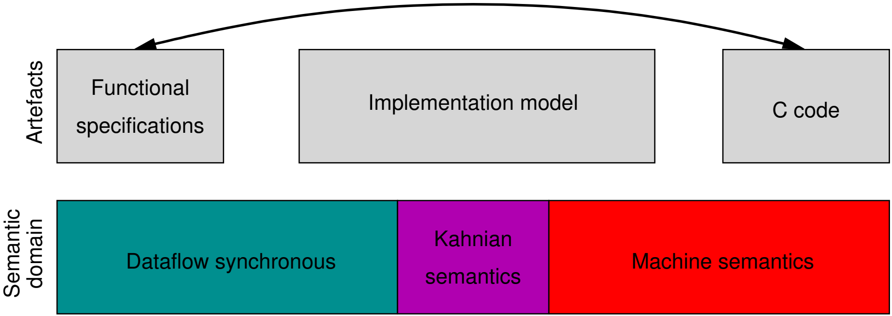
Dataflow representation of the implementation

Non-functional annotations

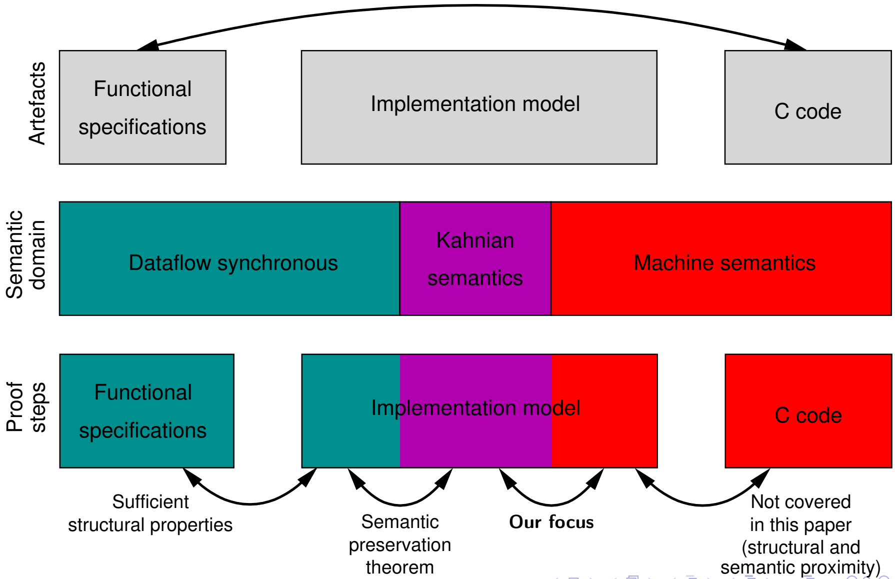
```
fun f:()->(float) at 0x20100
fun g:(int)->(int) at 0x30300
fun h:(float,int)->(int) at 0x20500
var
  x:int at 0x22000 y:float at 0x32000 z:int at 0x22004 d:int at z
  x0:int at x on cpu0 y0:float at y on cpu0 y1:float at y on cpu1
  z0:int at z on cpu0 d1:int at z on cpu1
  u:event at m1 u1:event at m1 v:event at m0 s0,s1,s2,s3,s4,s5,s6,s7,t0,t1,t2,t3:event
let
  d = 0 fby z
  u1 = top fby u
  s7 = top fby s6
thread on cpu0 at 0x20000 stack 0x30000
    wait(s7) done(s0)                        x0 = f()
    wait(s0) done(s1) [flush:0x22000]         x = x0

    wait(s1) done(s2) [lock:m0]               _ = v
    wait(s2) done(s3) [inval:0x32000] y0 = y
    wait(s3) done(s4)                        z0 = h(x0,y0)
    wait(s4) done(s5) [flush:0x22004]         z = z0
    wait(s5) done(s6) [unlock:m1]             u = top
thread on cpu1 at 0x30000 stack 0x40000
    done(t0)                   [lock:m1]             _ = u1
    wait(t0) done(t1) [inval:0x22004] d1 = d
    wait(t1) done(t2)                        y1 = g(d1)
    wait(t2) done(t3) [flush:0x32000]     y = y1
    wait(t3)                   [unlock:m0]          v = top
tel
```

- Machine semantics of memory coherency and synchronization operations

Proving the correctness of an implementation



Proving the correctness of an implementation



Correction formalization

From asynchronous to machine semantics

The proof is not complete, we merely provide a paper formalization of the proof objectives.

- **Implementability of the Kahnian interpretation**
  - Boundedness: Every FIFO of the Kahn network must be statically bounded for implementation in memory
  - Explicit synchronizations: Synchronization no longer on the data but exclusively on pure synchronization `event` variables
- **Mapping correctness**
  - Execution without errors: Execution under machine semantics do not lead to error state (static check of synchronization behavior)
  - Semantic preservation: Same sequence of inputs of function in Kahnian and machine semantics

Conclusion

Main claim : in practice, threaded implementations of dataflow specification preserve a fundamentally dataflow structure

- True for implementations we synthesize
- Future work : determine if it is true for other implementation methods

Impact on implementation correctness proof ?

- Proposal for new proof structuring (on paper)
- It is still difficult (future work)

No real-time yet

- Future work