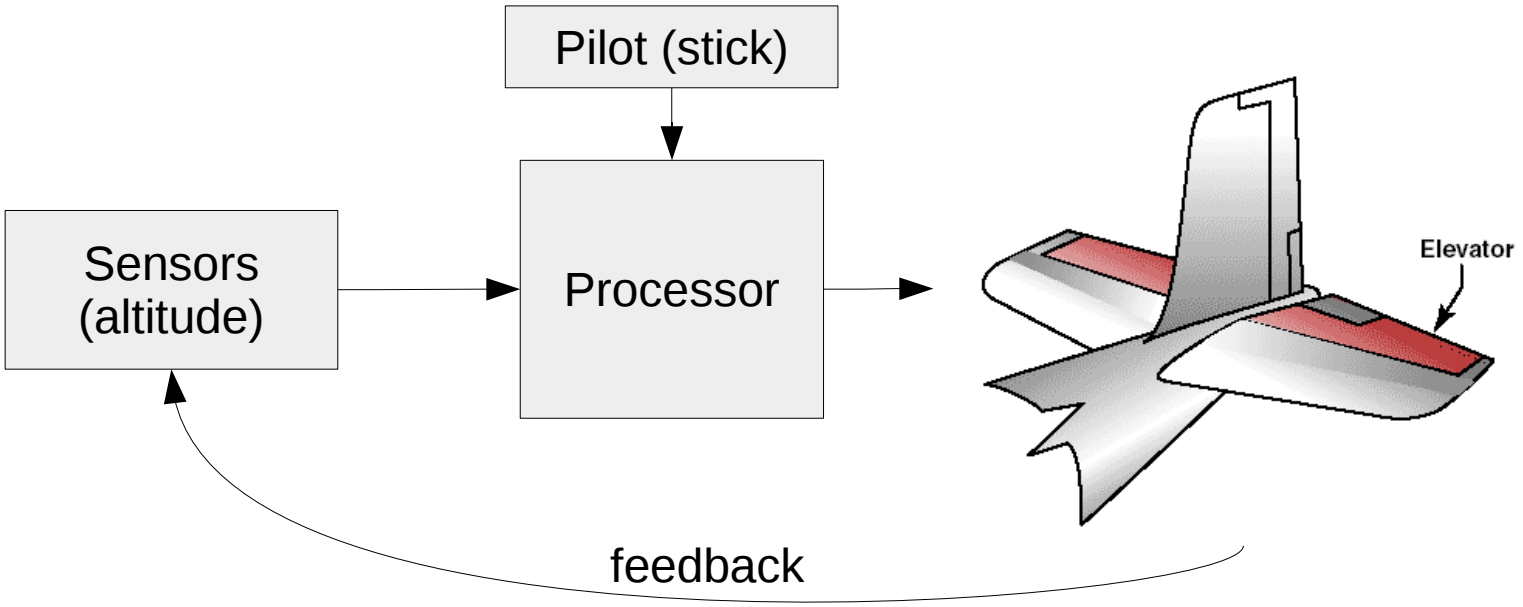
Code Generation of Time Critical Synchronous Programs on the Kalray MPPA Many-Core architecture

Amaury Graillat (PhD student)
Supervisors: Matthieu Moy, Pascal Raymond (Verimag)
Benoit Dinechin (Kalray)

Lundi 23 mai 2016, Télécom ParisTech

Safety Critical Systems

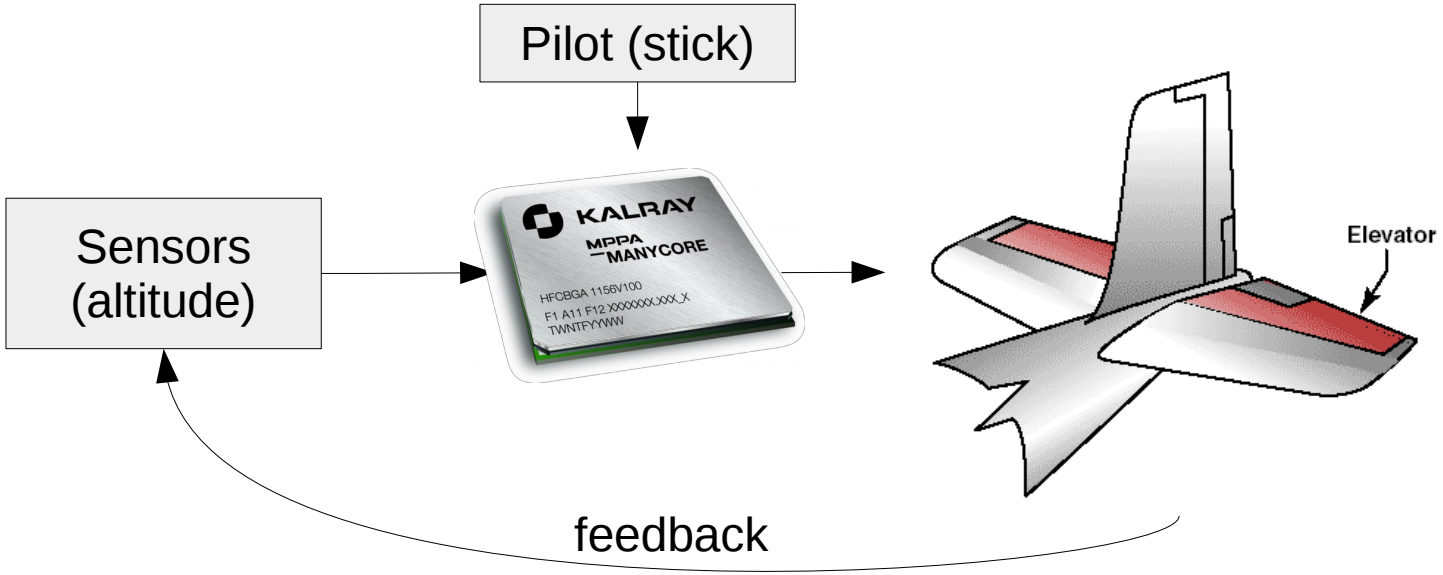
Example: an aircraft flight controller (control-command)



Time-critical: Latency sensor→actuator is part of the specification:
(Worst-Case Response Time)

Dataflow Sychronous languages: model of behavior with a notion of (logical) time
Research: Lustre, Heptagon, Esterel
Industrial: Esterel SCADE

Implementation of a Control Application on the Kalray MPPA



- **Model** written in Lustre/SCADE Synchronous Language
- Generate **parallel C code** for Kalray MPPA many-core architecture
- Preserve the semantics of the model

Case Study: ROSACE

Part of a flight controller (only altitude: actual systems have thousands nodes)
An open source case study from ONERA (http://sites.onera.fr/schedmcore/ROSACE)

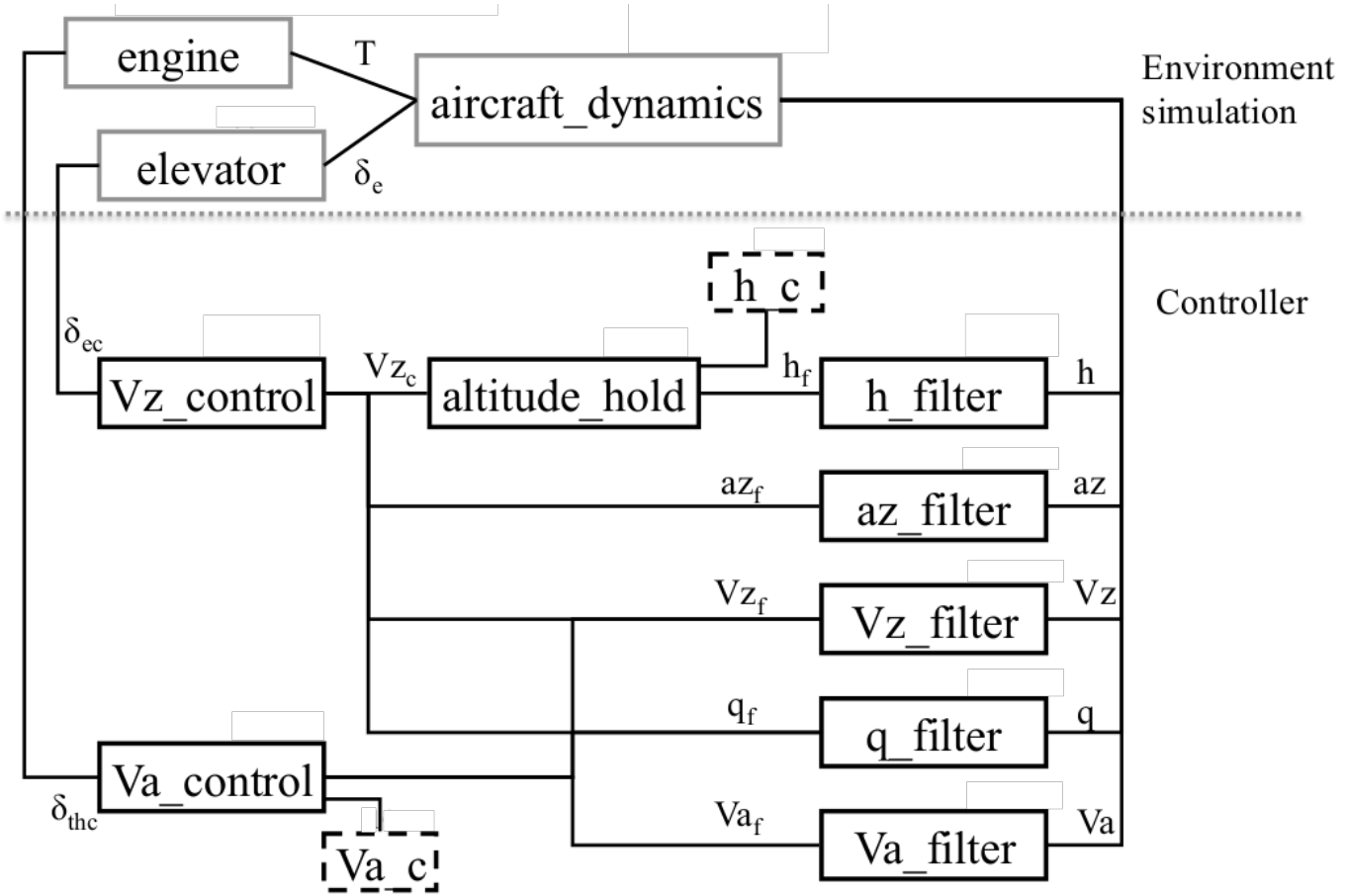


Image from [Pagetti et al., 2014]

Includes both physical simulation and controller.
This program is executed on a **specific period**.

Case Study: ROSACE

Part of a flight controller (only altitude: actual systems have thousands nodes)
An open source case study from ONERA (http://sites.onera.fr/schedmcore/ROSACE)

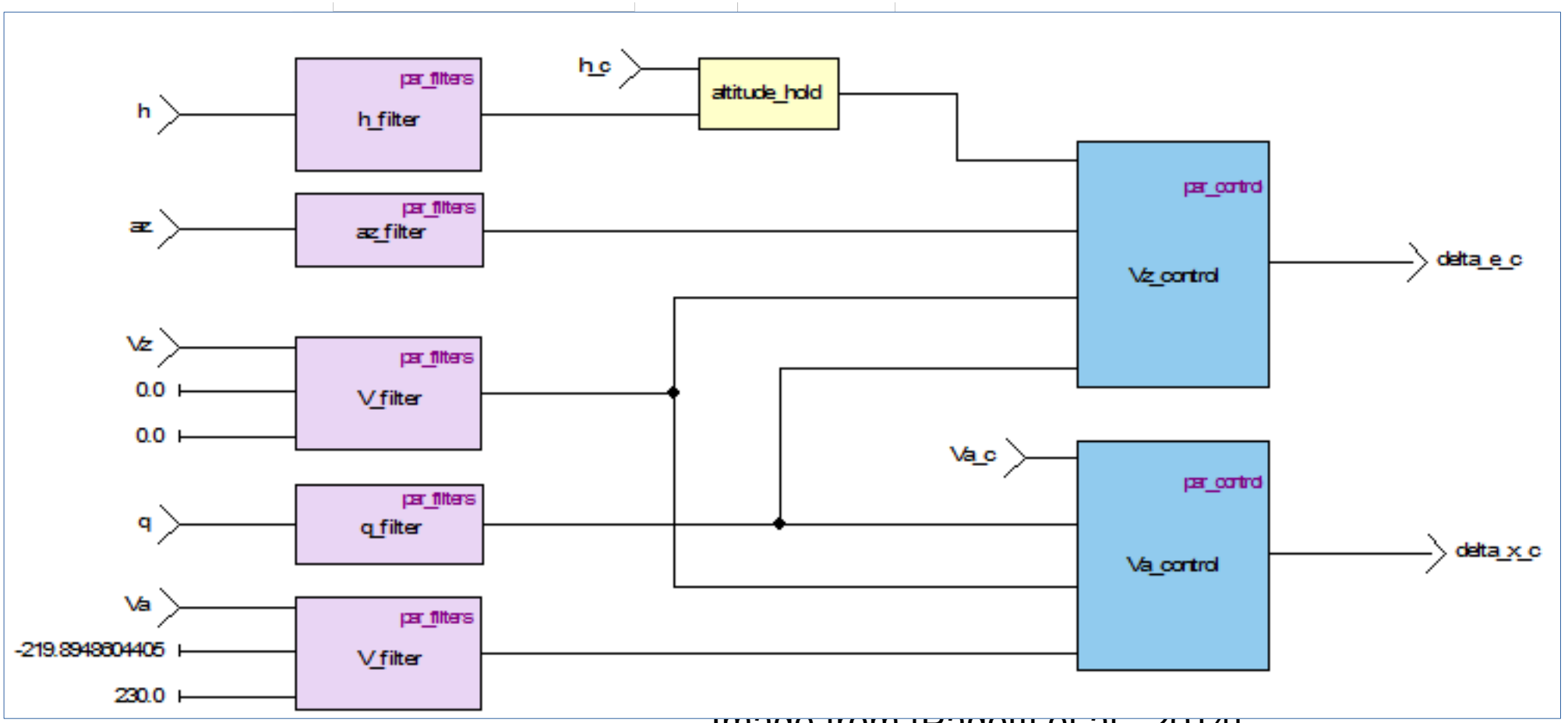
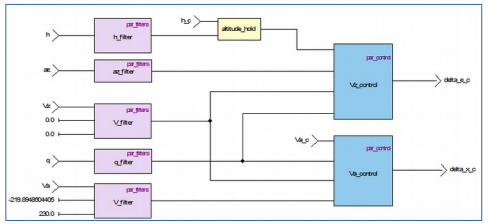


Image from [Pagetti et al., 2014]

Includes both physical simulation and controller.
This program is executed on a **specific period**.

“Traditional” Development Flow



Model of a program written in Dataflow Synchronous Language (SCADE, Lustre)

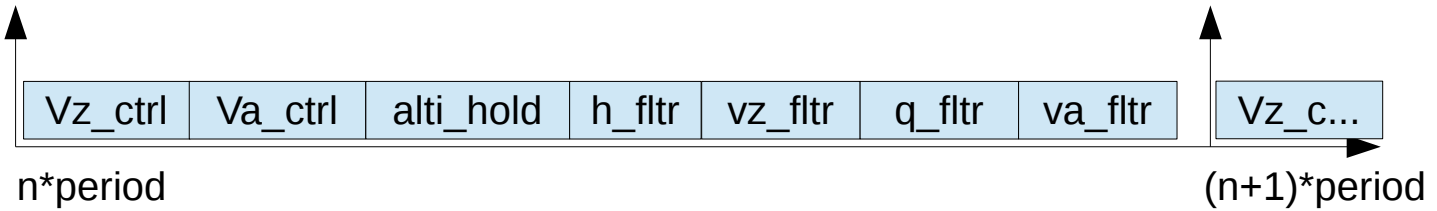
Lustre compiler / SCADE KCG

program.c

Single-core processor

```
while(true) {
      s = sensors()
      o = compute(s)
      actuators(o)
}
```

Worst-Case Response Time



WCRT < period

Single-, Multi-... Many-Core?

Single-core: become limited due to system complexity growth

Multi-core:

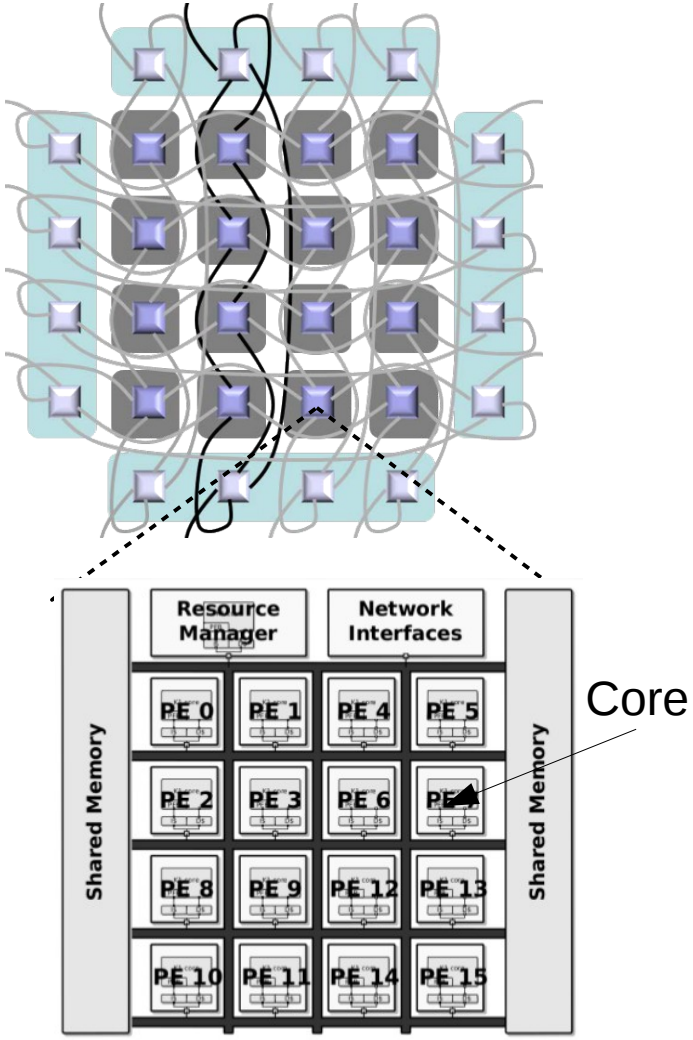
Shared memory

Timing anomalies (due to shared cache, branch prediction, etc) [Wilhelm et al., 2009]

… we need something else.

Many-Core: example of the Kalray MPPA

Common definition: several processors in the same chip communicating with a network.



Properties of the Kalray MPPA:

Cores:
- No complex branch prediction
- Only LRU caches

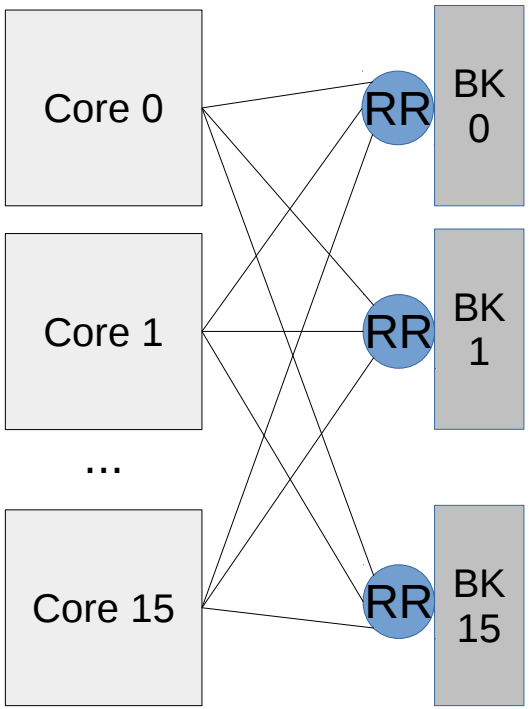
Cluster:
- Banked Shared-Memory (16*128ko)
- One round-robin for each bank access

Network-on-Chip to connect clusters:
- Bandwidth limiter
- (Network calculus possible)

In this talk, we are only considering one cluster.

Execution Model

Hardware: cluster



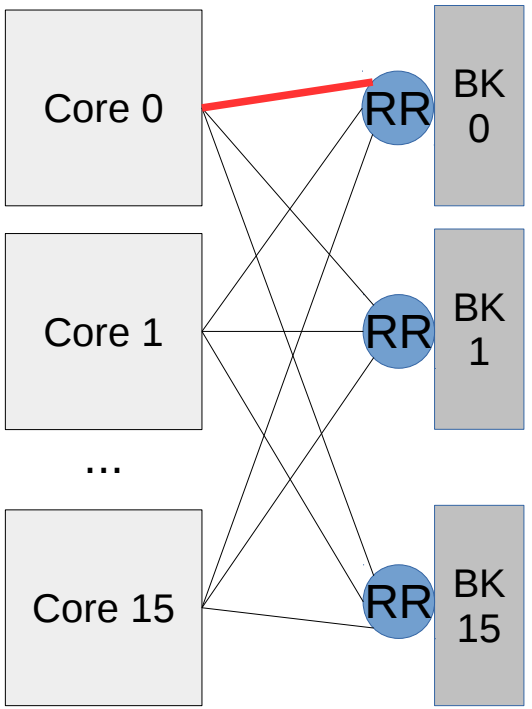
Every core has a private access to each memory bank round robin.

Software

- Parallelization of a program on one cluster
- Assign 1 core + 1 bank to each node
  - Code in private bank
  - Input data in private bank
- Static scheduling of nodes on each core

Execution Model

Hardware: cluster



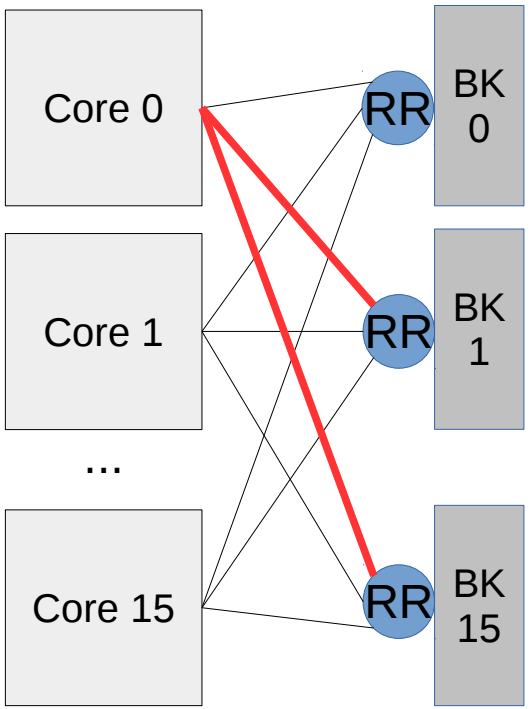
Every core has a private access to each memory bank round robin.

Software

- Parallelization of a program on one cluster
- Assign 1 core + 1 bank to each node
  - Code in private bank
  - Input data in private bank
- Static scheduling of nodes on each core

Execution Model

Hardware: cluster



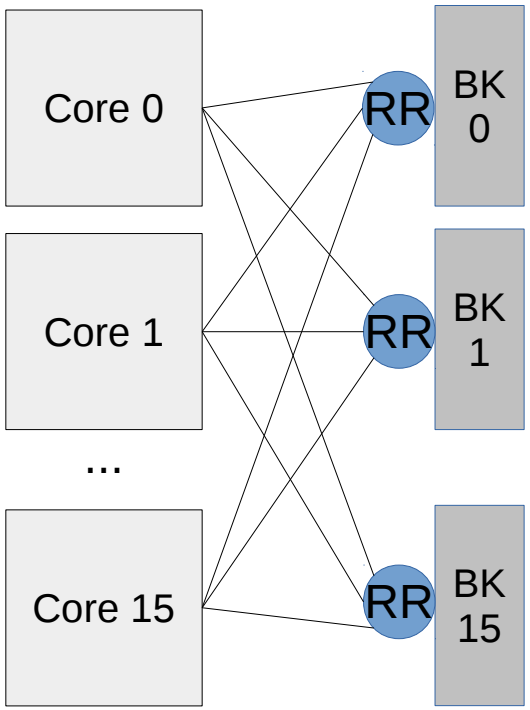
Every core has a private access to each memory bank round robin.

Software

- Parallelization of a program on one cluster
- Assign 1 core + 1 bank to each node
  - Code in private bank
  - Input data in private bank
- Static scheduling of nodes on each core

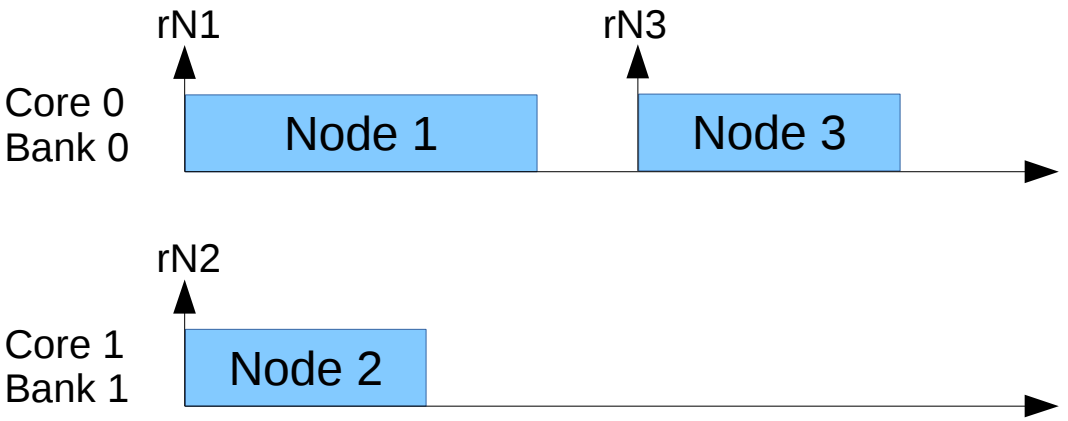
Execution Model

Hardware: cluster



Software

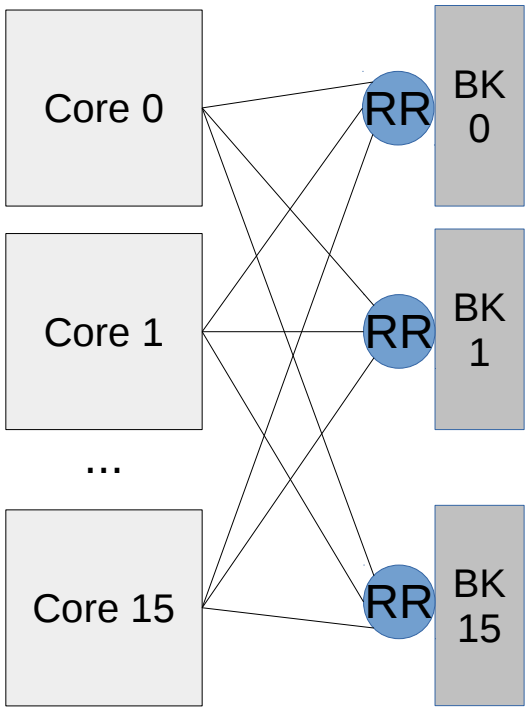
- Remote write policy
- Static release for computation: rN*
- Static release for write: rN*w



Hamza Rihani (Verimag) is working on interference bounding under these constraints

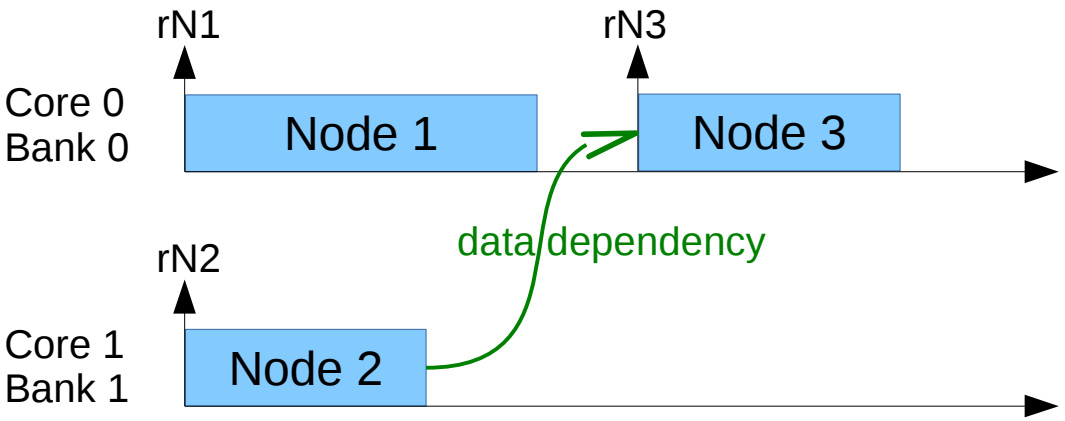
Execution Model

Hardware: cluster



Software

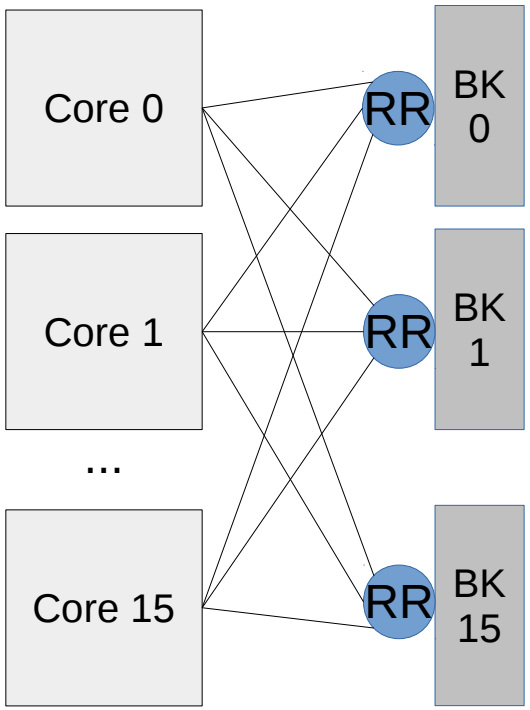
- Remote write policy
- Static release for computation: rN*
- Static release for write: rN*w



Hamza Rihani (Verimag) is working on interference bounding under these constraints

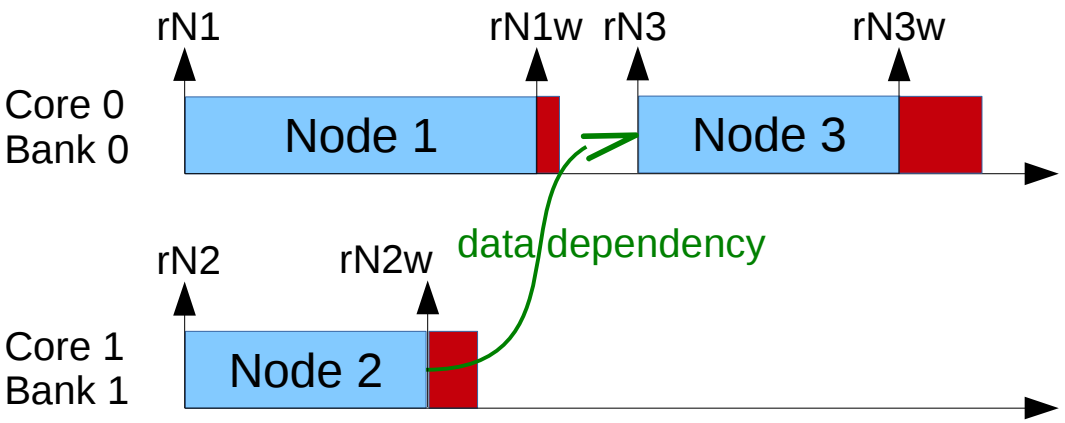
Execution Model

Hardware: cluster



Software

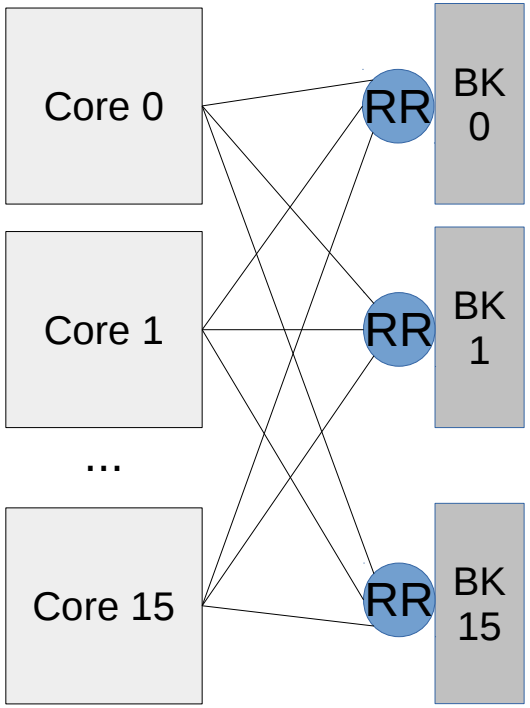
- Remote write policy
- Static release for computation: rN*
- Static release for write: rN*w



Hamza Rihani (Verimag) is working on interference bounding under these constraints

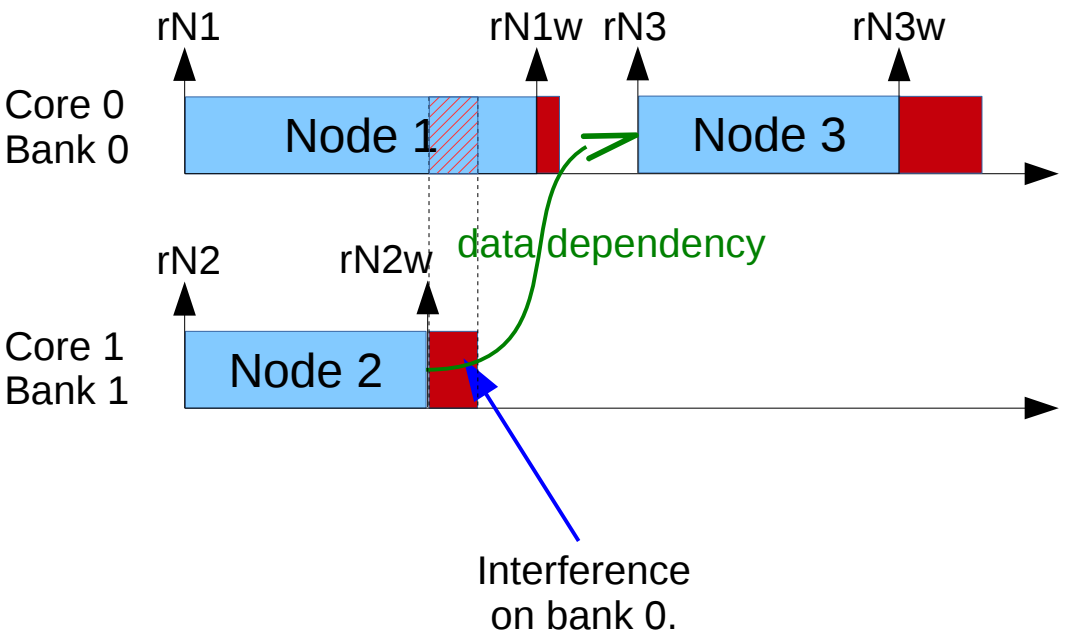
Execution Model

Hardware: cluster



Software

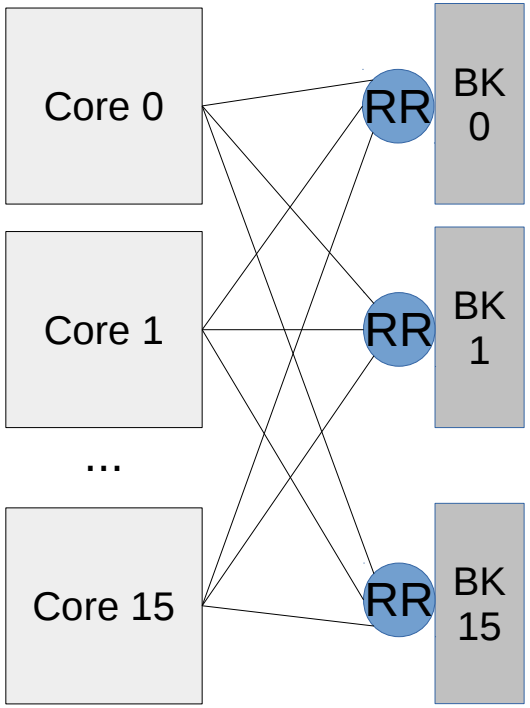
- Remote write policy
- Static release for computation: rN*
- Static release for write: rN*w



Hamza Rihani (Verimag) is working on interference bounding under these constraints

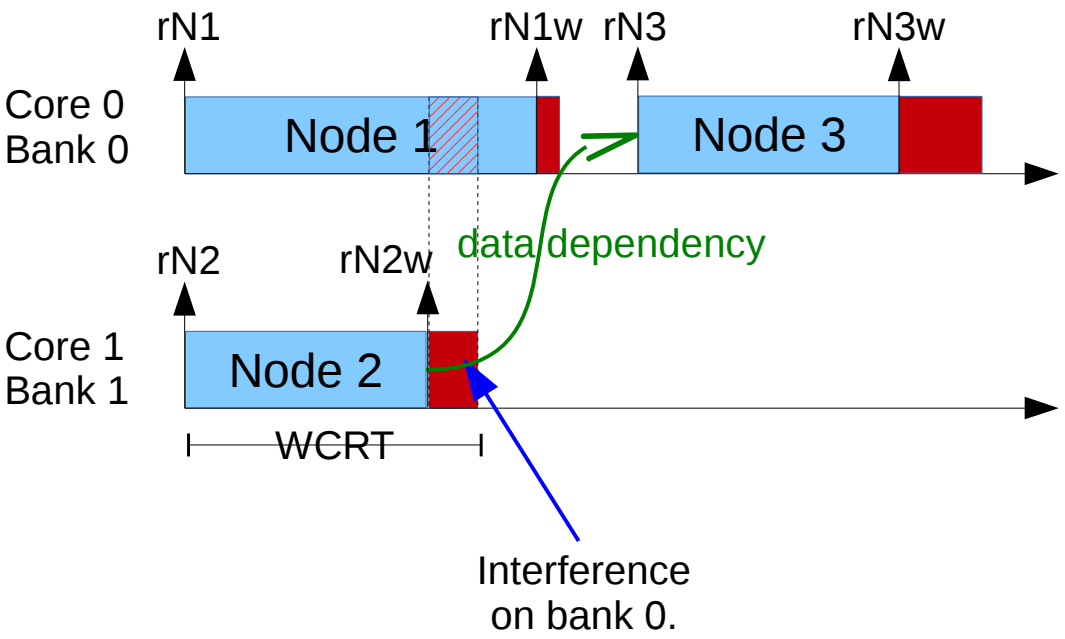
Execution Model

Hardware: cluster



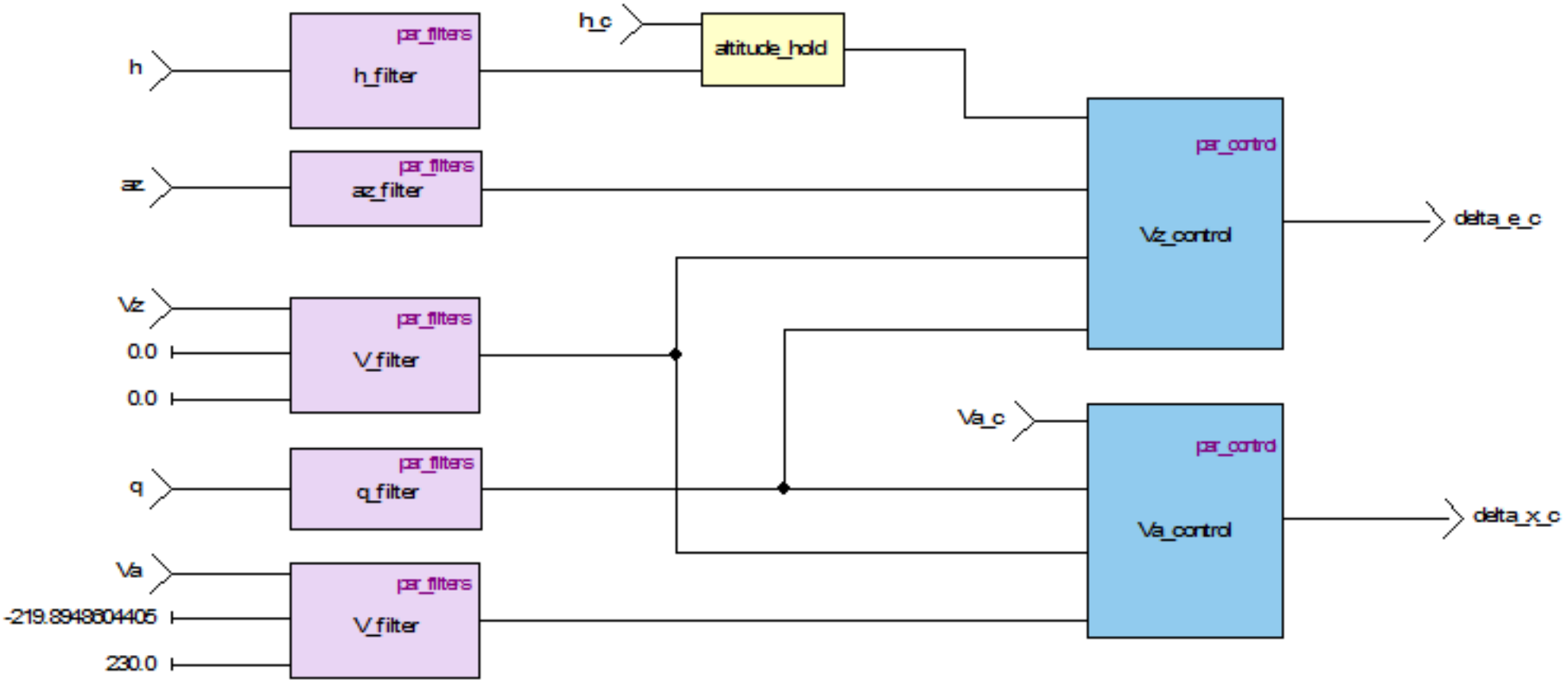
Software

- Remote write policy
- Static release for computation: rN*
- Static release for write: rN*w

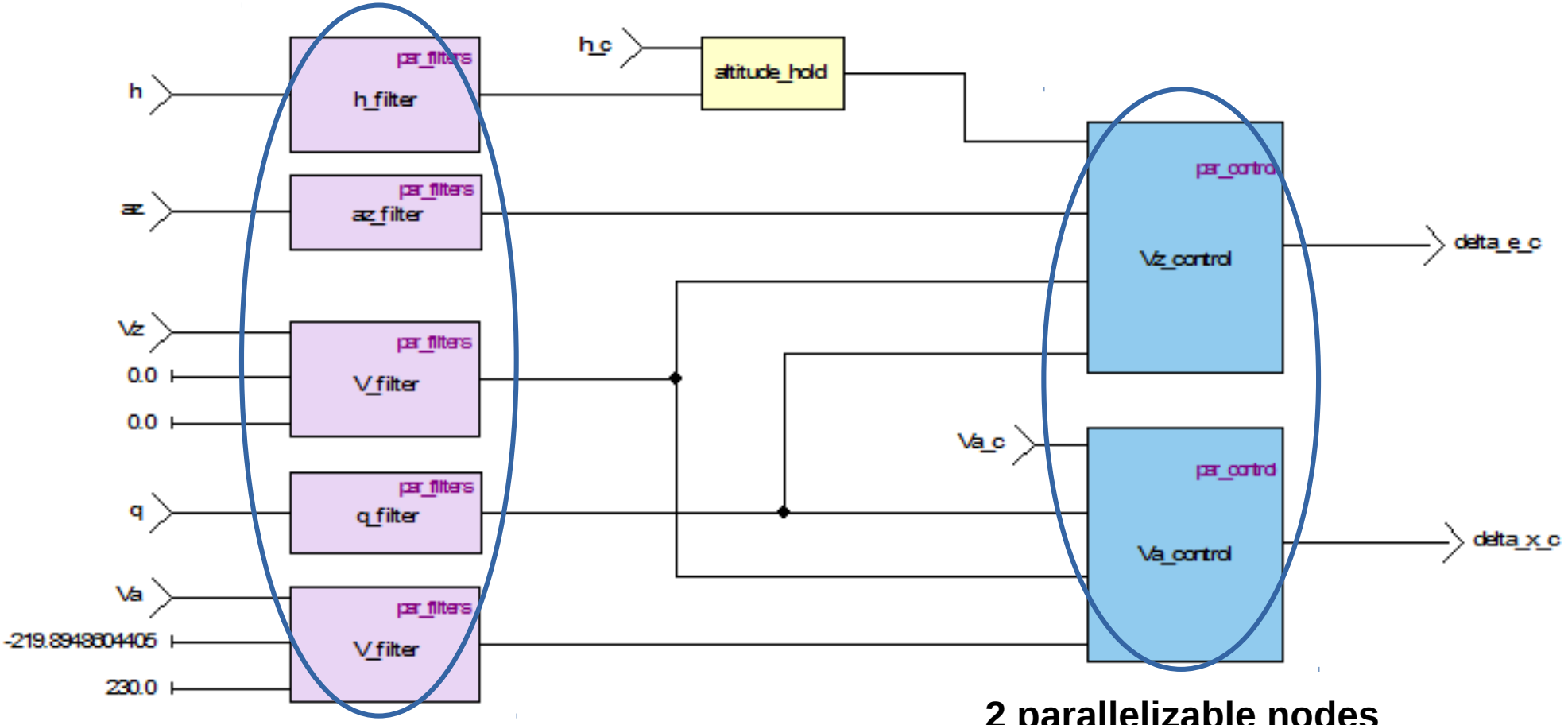


Hamza Rihani (Verimag) is working on interference bounding under these constraints

Parallelization of a Synchronous Dataflow Program



Parallelization of a Synchronous Dataflow Program



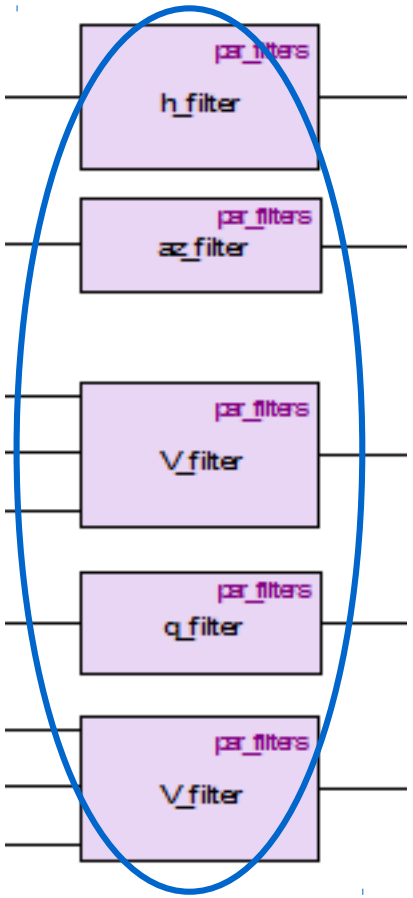
5 parallelizable nodes

2 parallelizable nodes

Code Generation

Implementation

Notation for parallelism in KCG Parallel prototype (partnership with Esterel)



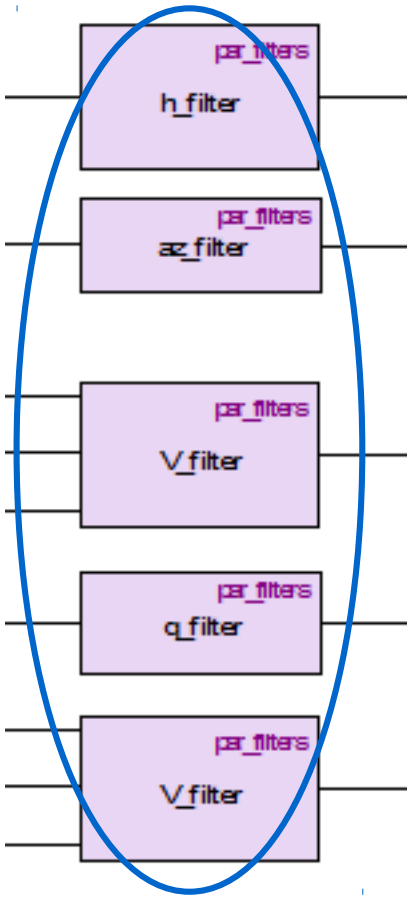

```
_L1 =                    q_filter(_L14);
_L2 =                    h_filter(_L11);
_L3 =                    V_filter(_L15, _L19, _L20);
_L4 =                    V_filter(_L13, _L16, _L17);
_L5 = altitude_hold(_L8, _L2);
_L6 =  Va_control(_L10, _L1, _L4, _L3);
_L8 = h_c;
_L9 =                    az_filter(_L12);

...

_L19 = -219.8948604405;
_L20 = 230.0;
_L22 = Vz_control_robust(_L5, _L9, _L4, _L1);
delta_x_c = _L6;
delta_e_c = _L22;
```

Semantics: all the nodes in the same group (#par_group) are execute in parallel.

Channels to communicate between groups
Special **macros** in c code: SEND inputs, RECV outputs

Implementation

Notation for parallelism in KCG Parallel prototype (partnership with Esterel)

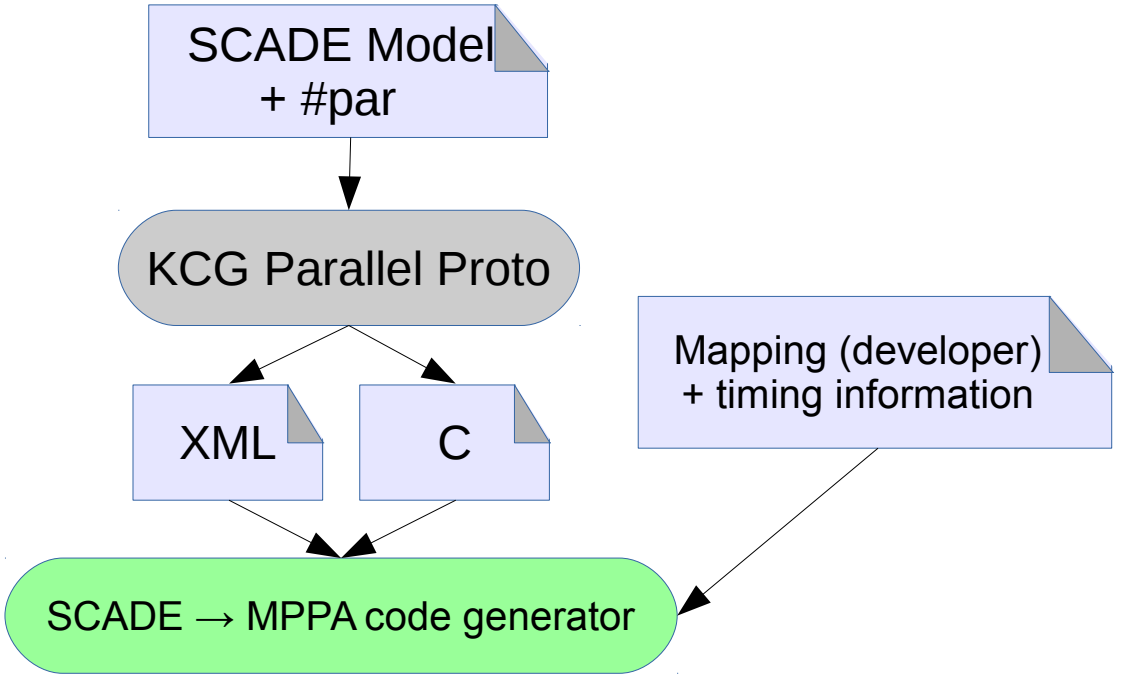



```
_L1 =  #par_filters q_filter(_L14);
_L2 =  #par_filters h_filter(_L11);
_L3 =  #par_filters V_filter(_L15, _L19, _L20);
_L4 =  #par_filters V_filter(_L13, _L16, _L17);
_L5 = altitude_hold(_L8, _L2);
_L6 =  Va_control(_L10, _L1, _L4, _L3);
_L8 = h_c;
_L9 =  #par_filters az_filter(_L12);

...

_L19 = -219.8948604405;
_L20 = 230.0;
_L22 = Vz_control_robust(_L5, _L9, _L4, _L1);
delta_x_c = _L6;
delta_e_c = _L22;
```

Semantics: all the nodes in the same group (#par_group) are execute in parallel.

Channels to communicate between groups
Special **macros** in c code: SEND inputs, RECV outputs

Implementation: General Flow



Implementation: User Mapping

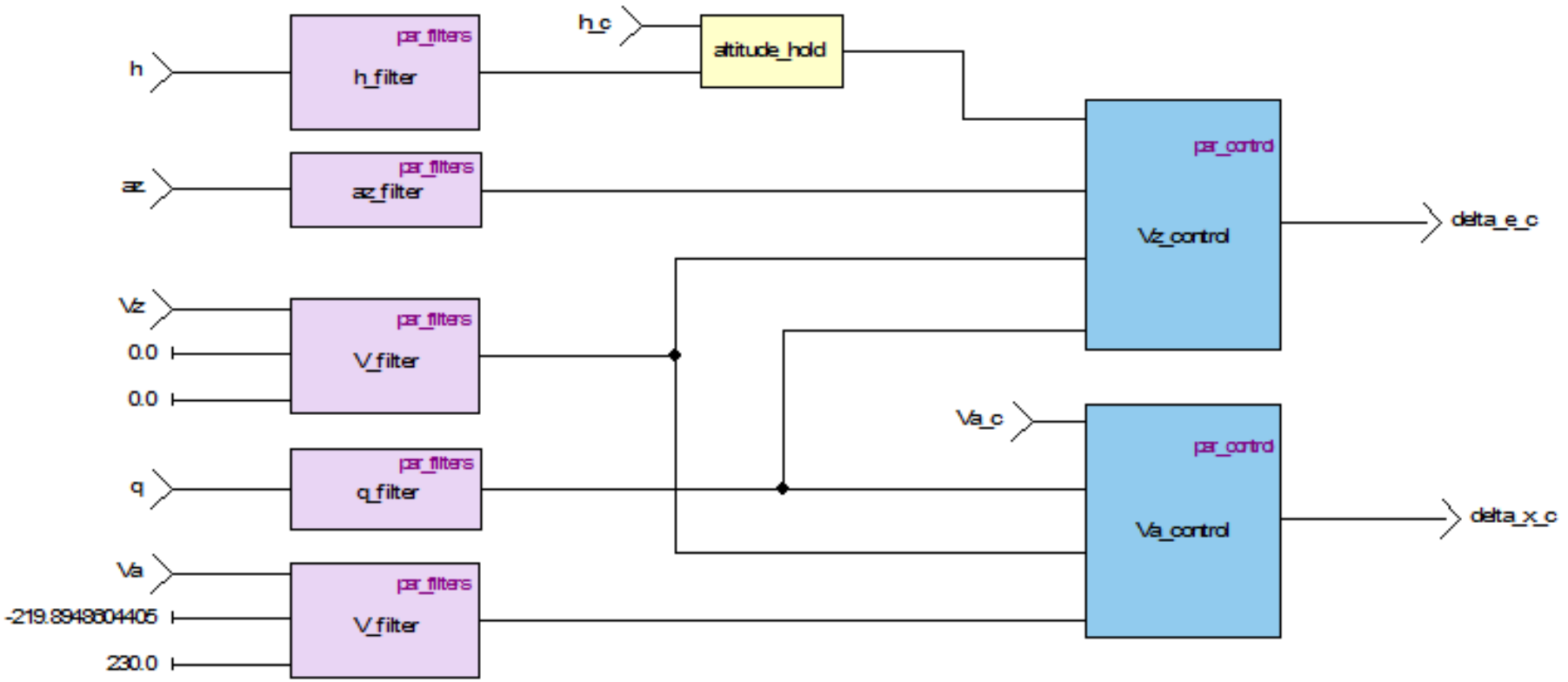
Runs on Core 0

Node name

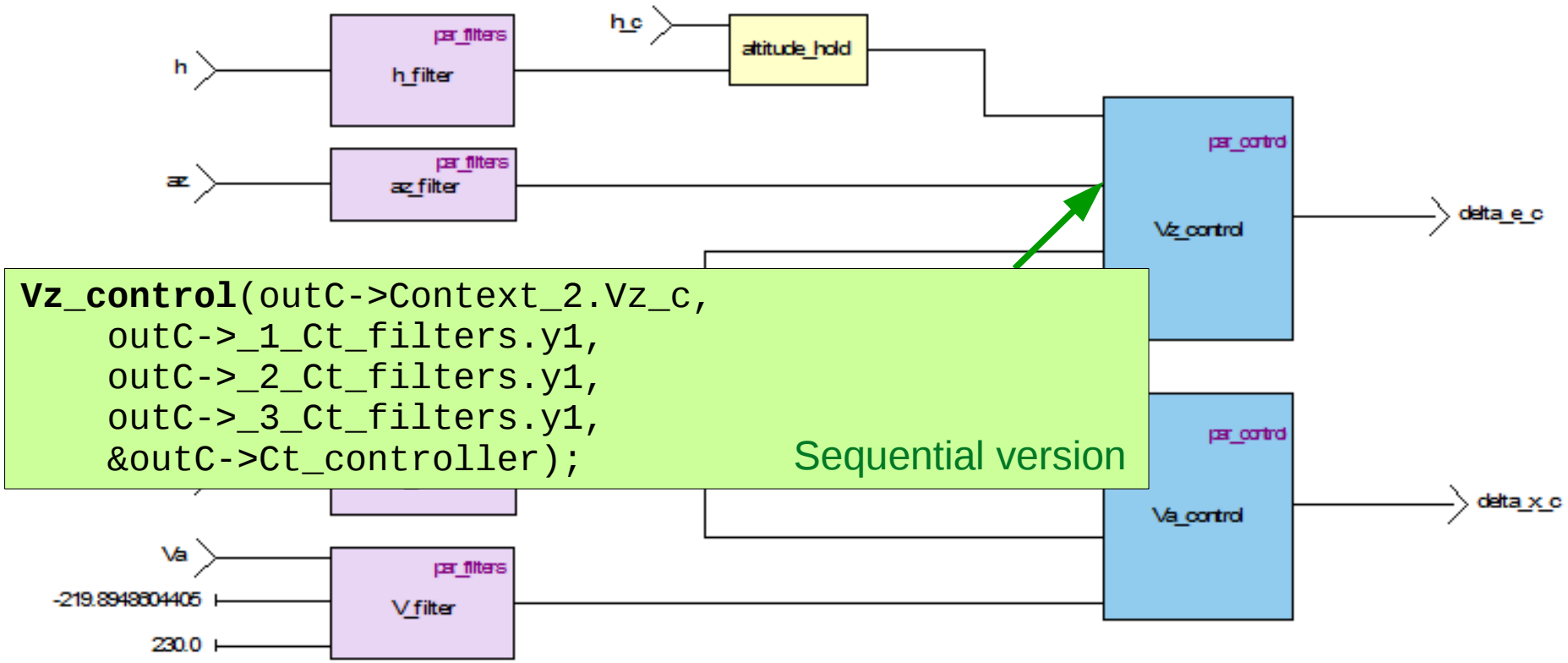
Display node output (debug)

```
<ressource core="0">
    <node name="RosaceSimul" />
</ressource>
<ressource core="1">
    <node name="V_filter" />
</ressource>
<ressource core="2">
    <node name="V_filter"
        worker="_2_V_filter_worker" />
</ressource>
<ressource core="3">
    <node name="q_filter" />
    <node monitor="1" name="Va_control" />
</ressource>
<ressource core="4">
    <node name="az_filter" />
    <node name="Vz_control" />
</ressource>
<ressource core="5">
    <node name="h_filter" />
</ressource>
```

Node Call (Generated Macros)



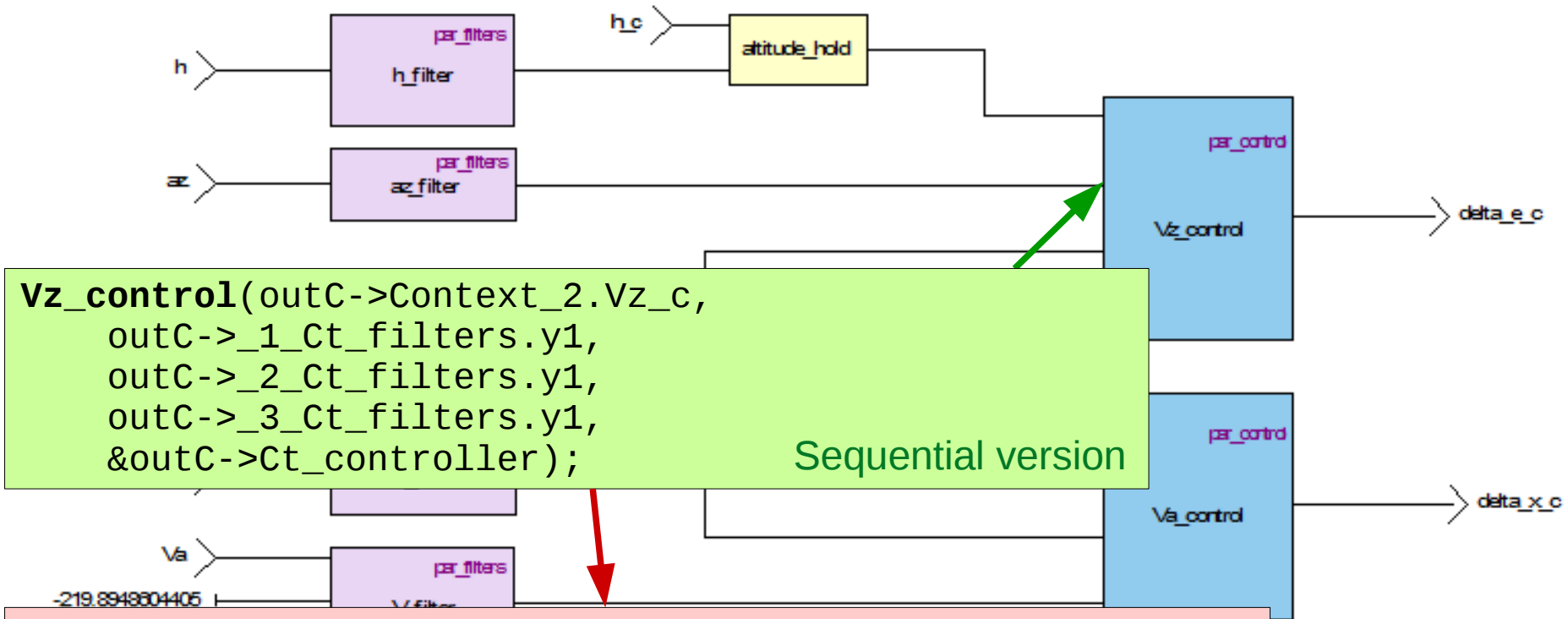
Node Call (Generated Macros)




```
Vz_control(outC->Context_2.Vz_c,
    outC->_1_Ct_filters.y1,
    outC->_2_Ct_filters.y1,
    outC->_3_Ct_filters.y1,
    &outC->Ct_controller);
```

Sequential version

Node Call (Generated Macros)



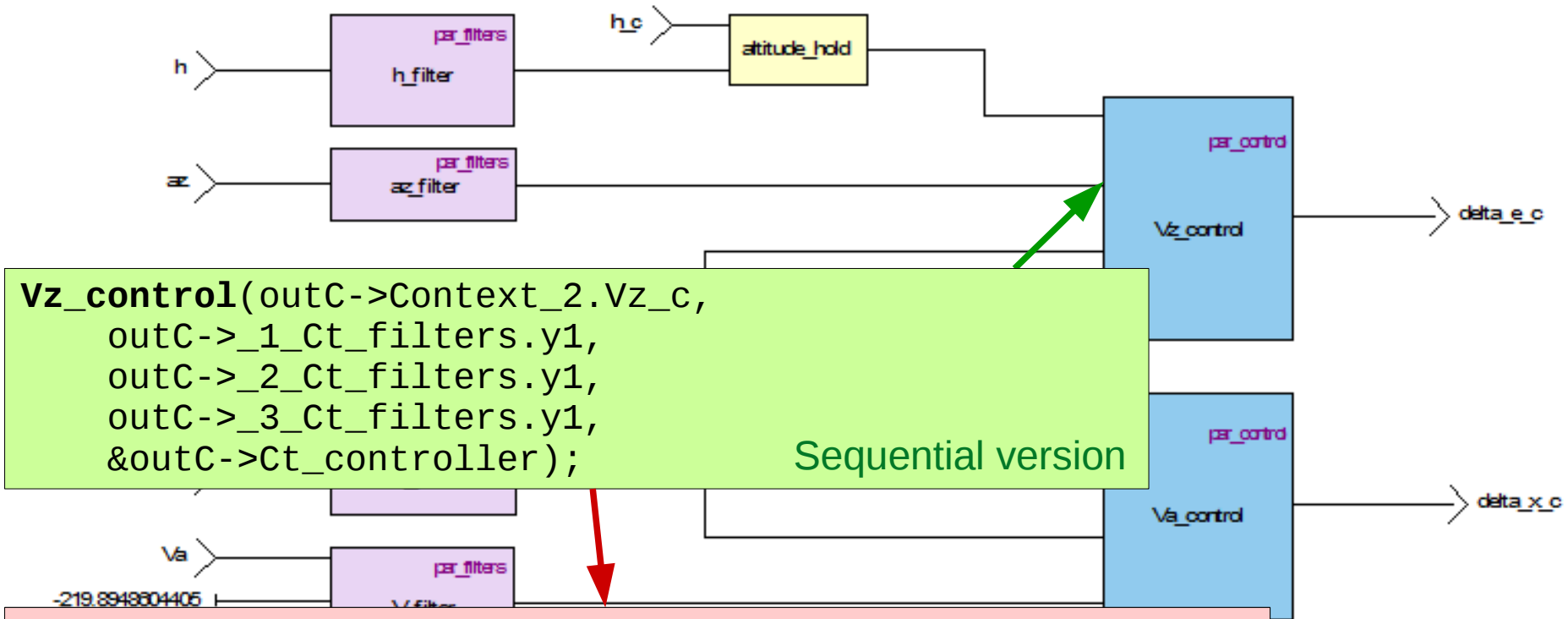

```
Vz_control(outC->Context_2.Vz_c,
    outC->_1_Ct_filters.y1,
    outC->_2_Ct_filters.y1,
    outC->_3_Ct_filters.y1,
    &outC->Ct_controller);
```

Sequential version

```
// Write data in the channel
// Send data
...
// Read result
```

Parallel version

Node Call (Generated Macros)



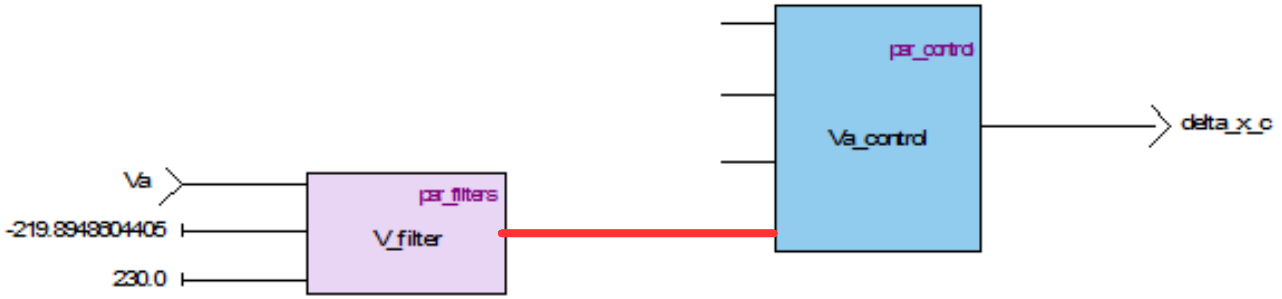

```
Vz_control(outC->Context_2.Vz_c,
    outC->_1_Ct_filters.y1,
    outC->_2_Ct_filters.y1,
    outC->_3_Ct_filters.y1,
    &outC->Ct_controller);
```

Sequential version

```
Vz_control_in_ch_sub.Vz_c = outC->Context_altitude_hold_2.Vz_c;
KCG_CHANNEL_SEND_Vz_control_in_ch_sub(Vz_control_in_ch_sub);
...
KCG_CHANNEL_RECV_Vz_control_out_ch(Vz_control_out_ch);
```

Parallel version

Generated Wrapper



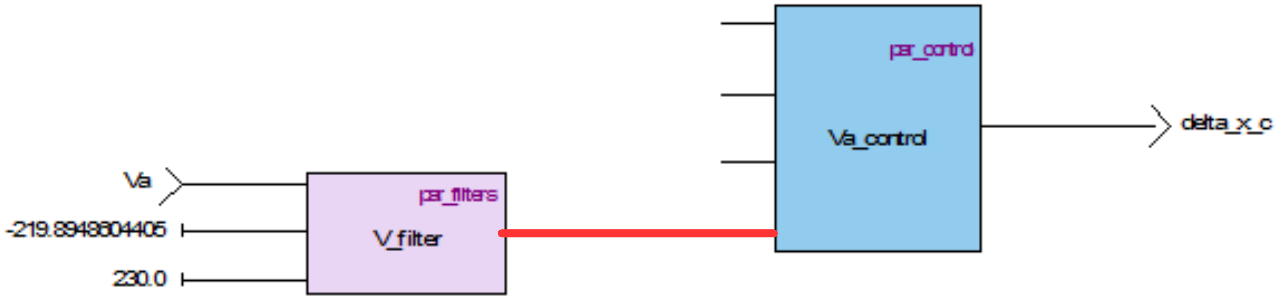

```
void V_filter_worker(outC_V_filter_worker *outC)
{
  kcg_float64 y1_0, x1_0, in1;
```

Read inputs

Compute **V_filter**

Write result to **Va_control**

Generated Wrapper



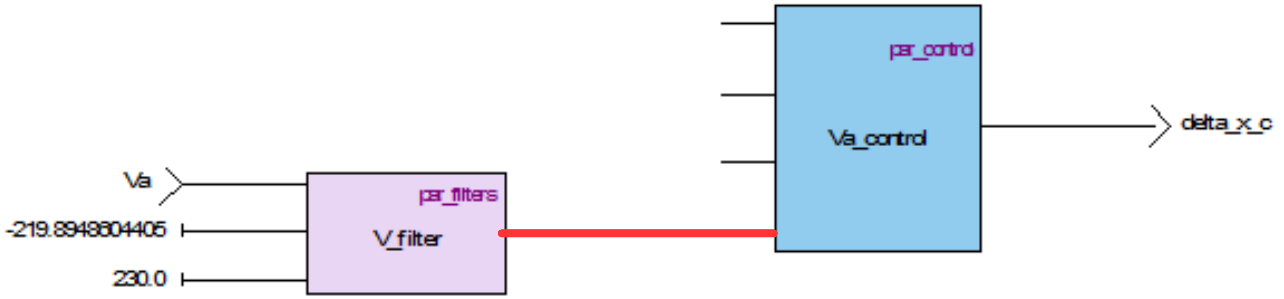

```
void V_filter_worker(outC_V_filter_worker *outC)
{
  kcg_float64 y1_0, x1_0, in1;
```

Read inputs

```
  V_filter(in1,x1_0,y1_0,&_6_V_filter_out_ch_sub.y1,&outC->Context_V_filter);
```

Write result to **Va_control**

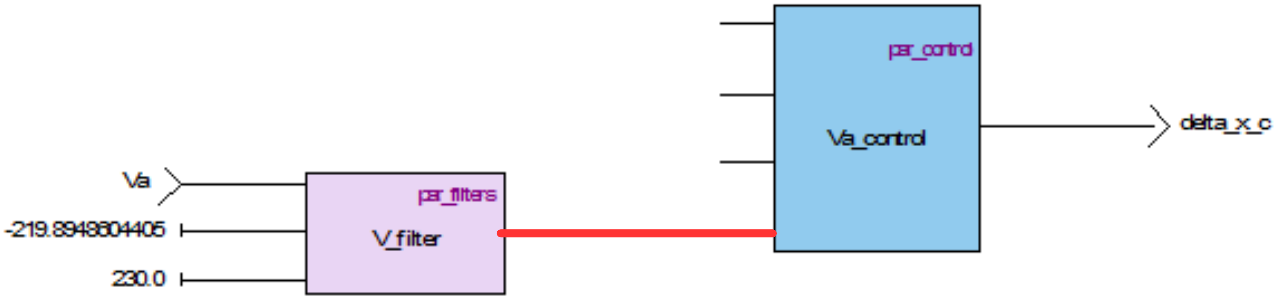
Generated Wrapper




```
void V_filter_worker(outC_V_filter_worker *outC)
{
  kcg_float64 y1_0, x1_0, in1;

  KCG_RECV_V_filter_in_ch(V_filter_in_ch);
  in1 = V_filter_in_ch.in1;
  x1_0 = V_filter_in_ch.x1_0;
  y1_0 = V_filter_in_ch.y1_0;


  V_filter(in1,x1_0,y1_0,&_6_V_filter_out_ch_sub.y1,&outC->Context_V_filter);
```

Write result to **Va_control**

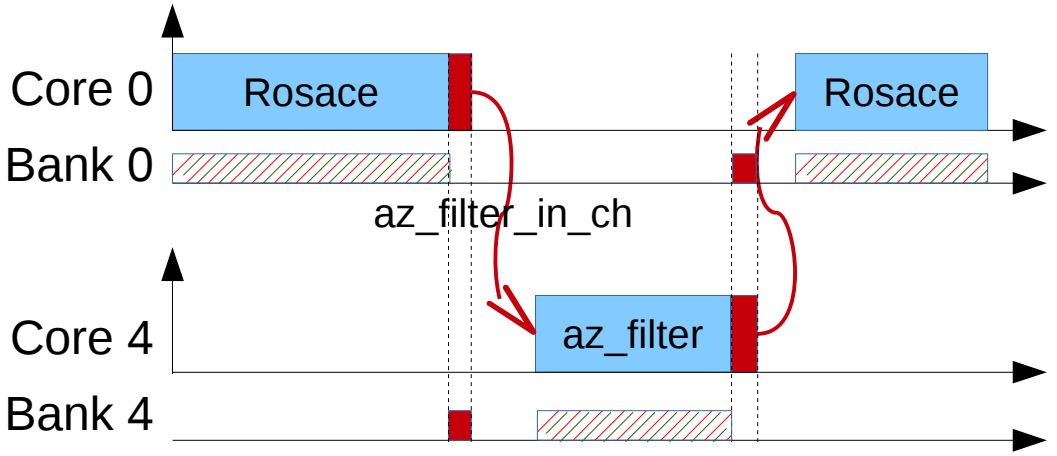
Generated Wrapper




```
void V_filter_worker(outC_V_filter_worker *outC)
{
  kcg_float64 y1_0, x1_0, in1;

  KCG_RECV_V_filter_in_ch(V_filter_in_ch);
  in1 = V_filter_in_ch.in1;
  x1_0 = V_filter_in_ch.x1_0;
  y1_0 = V_filter_in_ch.y1_0;


  V_filter(in1,x1_0,y1_0,&_6_V_filter_out_ch_sub.y1,&outC->Context_V_filter);


  KCG_SEND__6_V_filter_out_ch_sub(_6_V_filter_out_ch_sub);
  _5_V_filter_out_ch_sub.y1 = _6_V_filter_out_ch_sub.y1;
  KCG_SEND__5_V_filter_out_ch_sub(_5_V_filter_out_ch_sub);
}
```

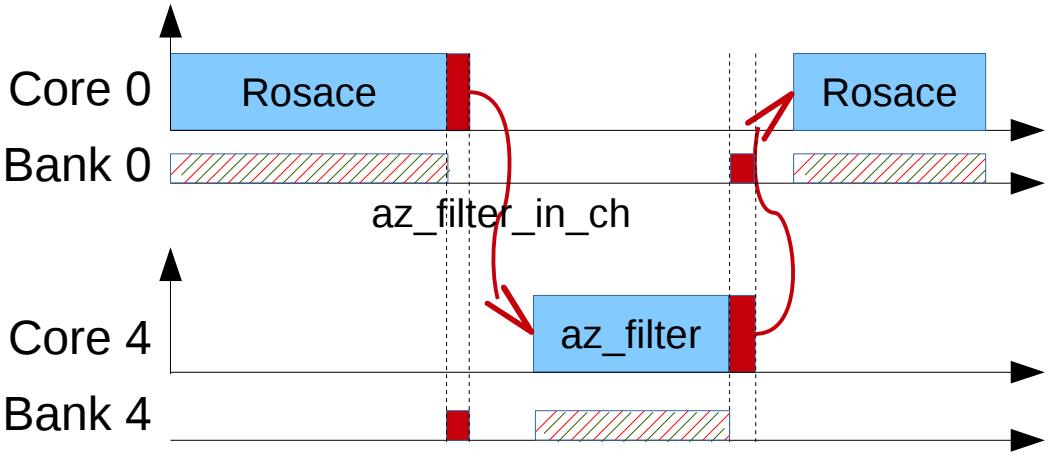
Shared-Memory Communications

MPPA core caches are non-coherent: need flush and memory barriers.
Channel structure is locate in the destination memory (contains validity token)



Shared-Memory Communications

MPPA core caches are non-coherent: need flush and memory barriers.
Channel structure is locate in the destination memory (contains validity token)



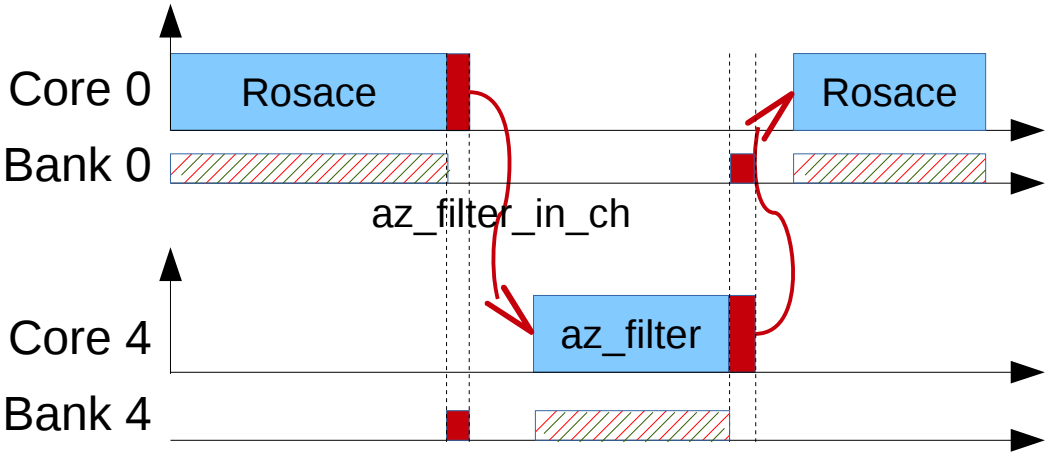

```
#define KCG_CHANNEL_SEND_az_filter_in_ch(c) {
    MEM_WRITE_PURGE;
    MEM_BARR;

    az_filter_in_ch.data=true;
    MEM_WRITE_PURGE;
    MEM_BARR;
}
```

Make sure data has been written in Mem 4 (purge write buffer)

Push token in Mem 4 (purge write buffer)

Shared-Memory Communications

MPPA core caches are non-coherent: need flush and memory barriers.
Channel structure is locate in the destination memory (contains validity token)




```
#define KCG_CHANNEL_SEND_az_filter_in_ch(c) {
    MEM_WRITE_PURGE;
    MEM_BARR;
```

Make sure data has been written in Mem 4 (purge write buffer)

```
    az_filter_in_ch.data=true;
    MEM_WRITE_PURGE;
    MEM_BARR;
}
```

Push token in Mem 4 (purge write buffer)

```
 #define KCG_CHANNEL_RECV_az_filter_in_ch(c) {
    while((!BYPASS_CACHE(&az_filter_in_ch.data)));
```

Polling on token

```
    MEM_READ_PURGE;
    MEM_BARR;
```

Flush data cache (force refresh data)

```
    az_filter_in_ch.data=false;
    MEM_WRITE_PURGE;
    MEM_BARR;
 }
```

Reset token

Tasks Creation

Launch the workers on the cores.

```
if(utask_start_pe(&thPE3, NULL, thread_PE3, NULL, PE3))
{ return -1; }
```

Tasks Creation

Launch the workers on the cores.

```
if(utask_start_pe(&thPE3, NULL, thread_PE3, NULL, PE3))
{ return -1; }
```

Static scheduling on cores.

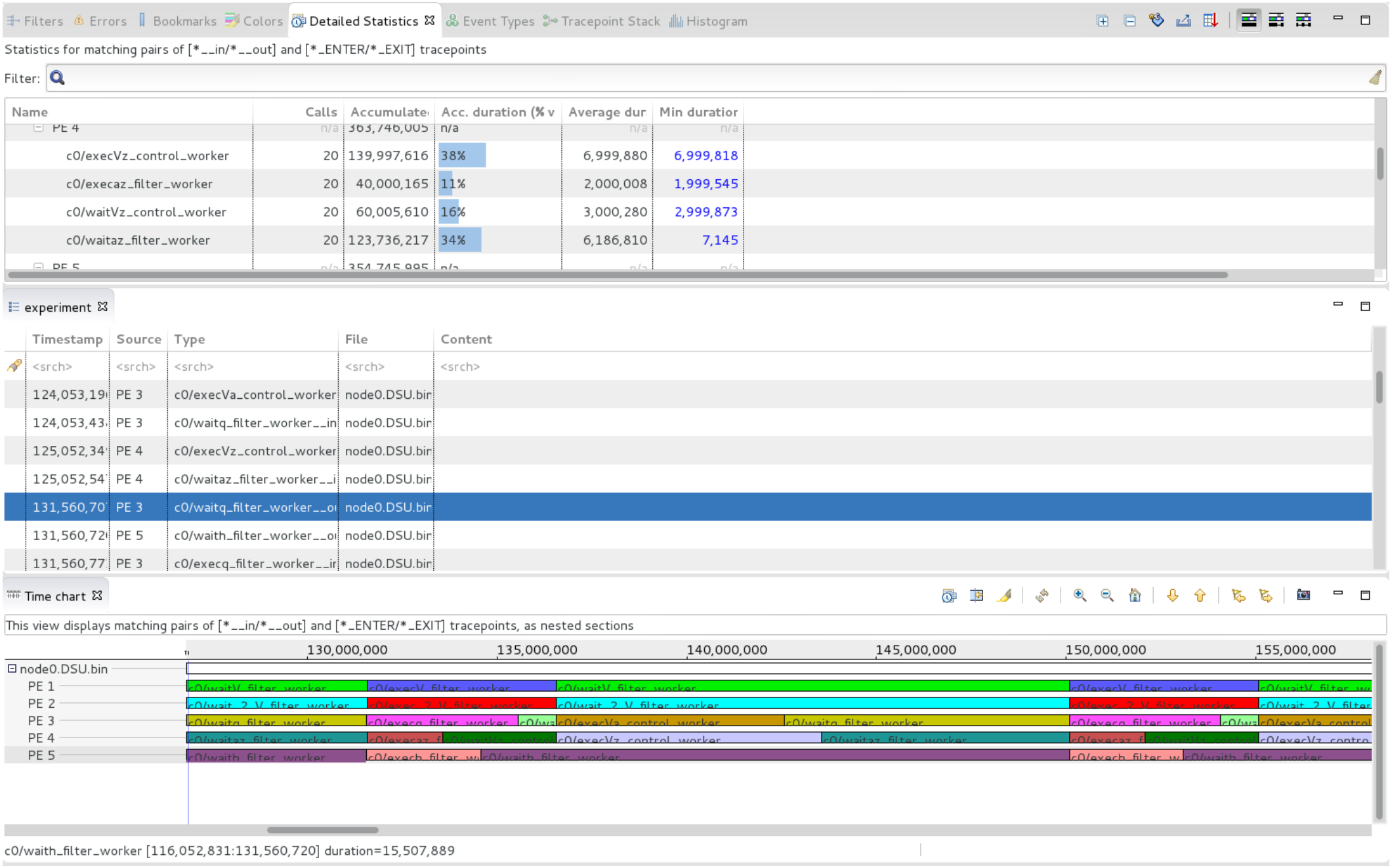
```
void *thread_PE3(__attribute__((__unused__)) void *args) {

    q_filter_worker_init(&q_filter_worker_outC);
    Va_control_worker_init(&Va_control_worker_outC);

    for(int loop=0; loop<NB_STEPS; loop++) {
        WAIT_UNTIL(loop*PERIOD);

        q_filter_worker(&q_filter_worker_outC);
        Va_control_worker(&Va_control_worker_outC);

    }
    return NULL;
}
```

Initialize the node instances

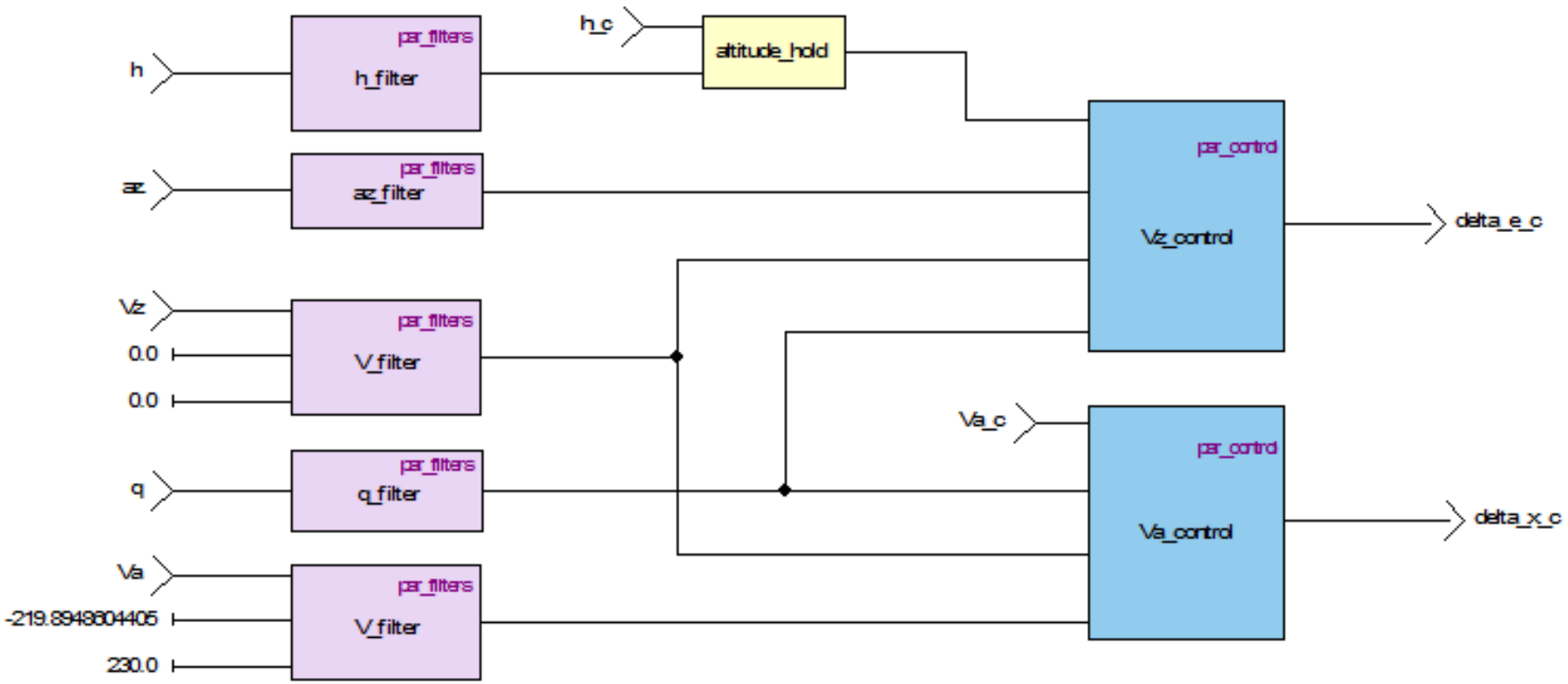
Run the wrappers

Performance Evaluation

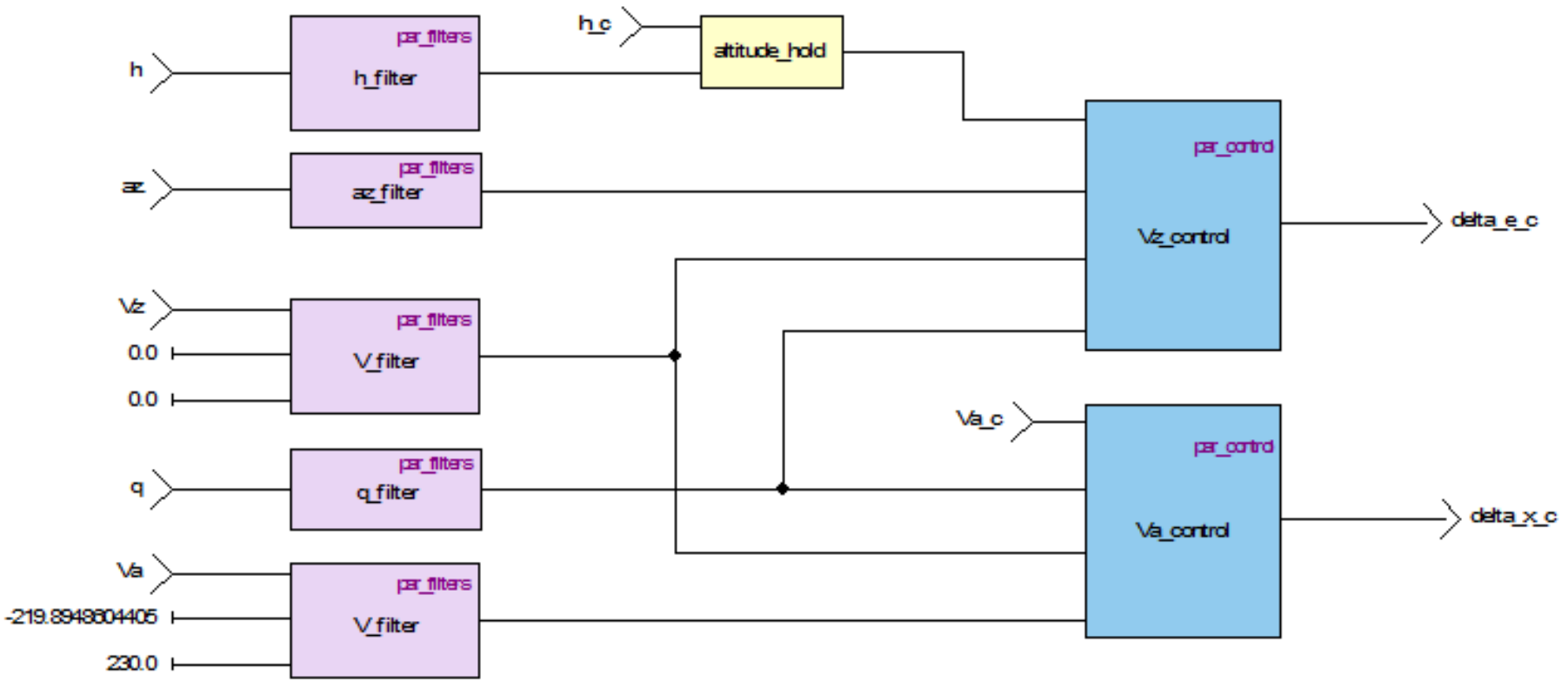
Execution Traces

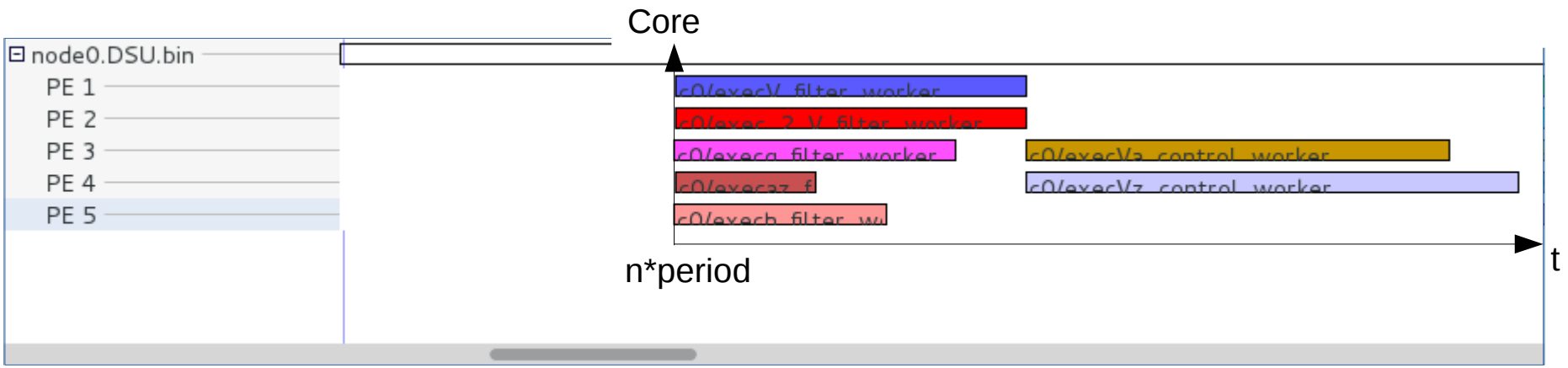


Execution Traces

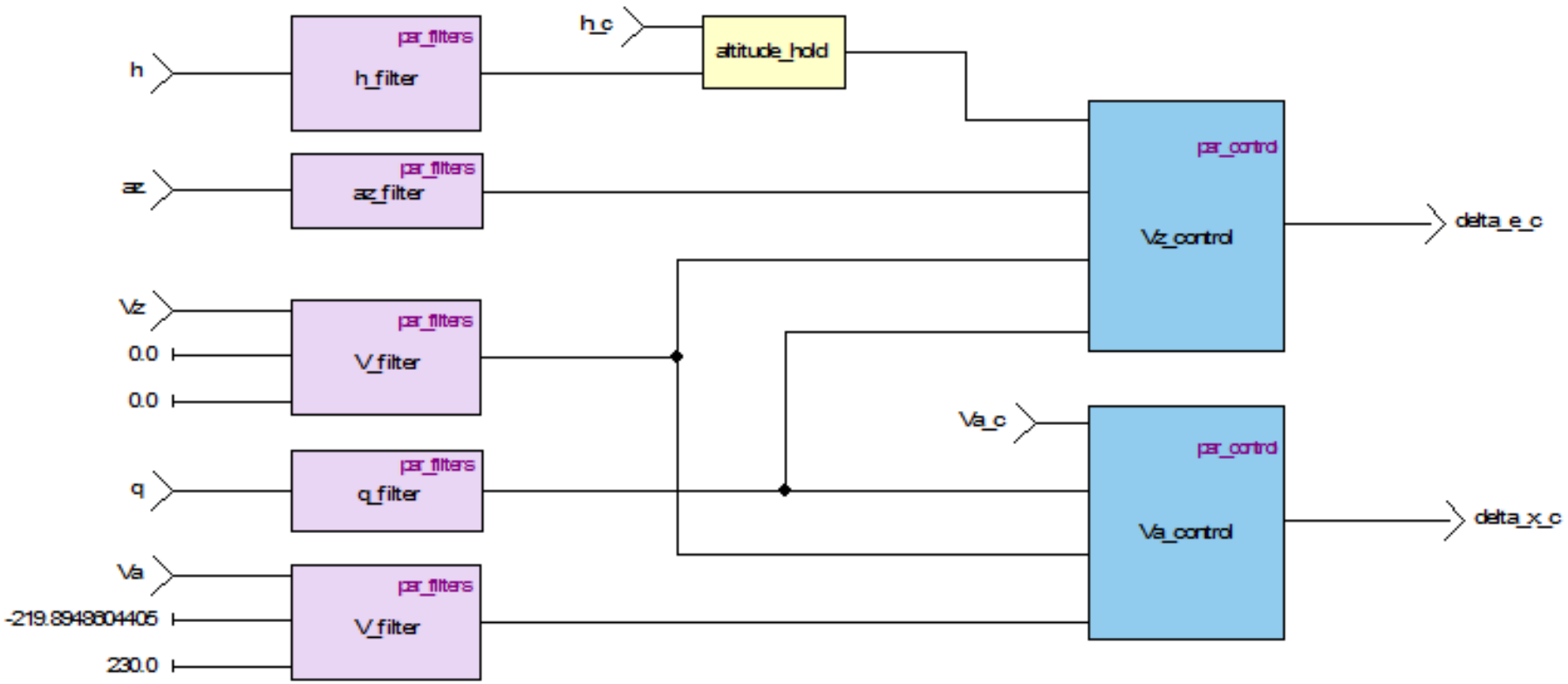


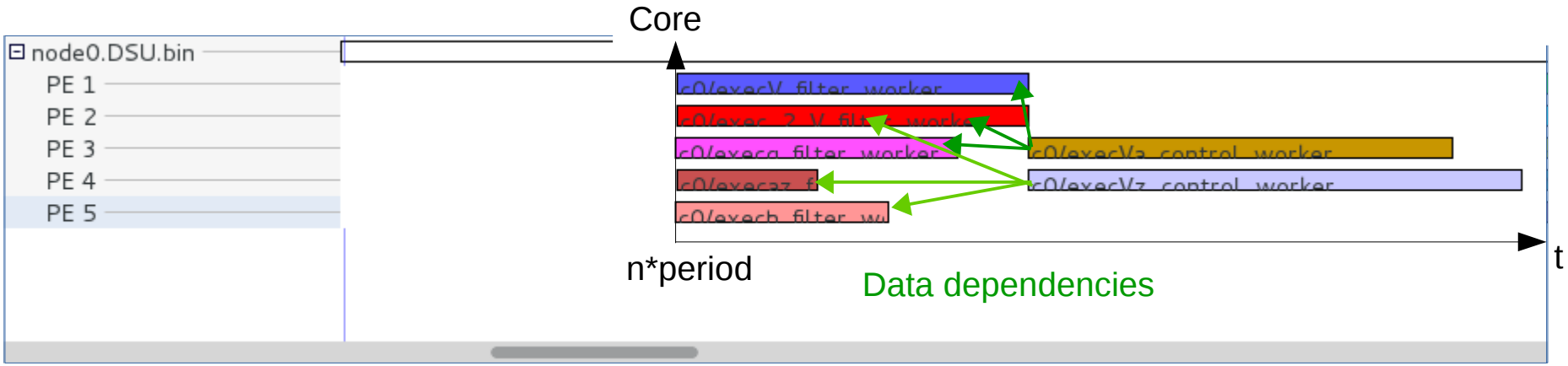
Execution Traces





Execution Traces





Communication Overhead

- **Cost of Memory access**
  8 cycles per 64 bytes of data
- **Cost of Write Buffer flush**
  8 entries cache = 8 memory accesses
- **Data Cache flush**
  Costs nothing (but pays the memory access for each miss)

Conclusion

- Model-base code generation for a Many-core
- Semantics preserving approach
- Time-critical constraints

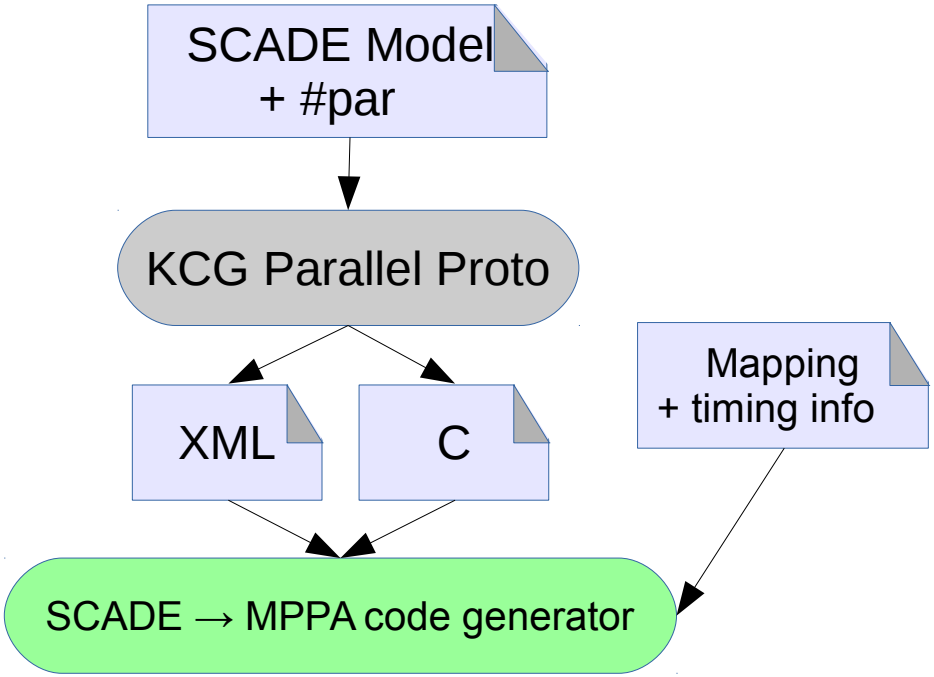
Future Improvements

- Multi-clusters
- Multi-periodic programs

Code Generation of Time Critical Synchronous Programs
on the Kalray MPPA Many-Core architecture

Thank you for you attention.

Any question?



Amaury Graillat (PhD student)
amaury.graillat@imag.fr