PREESM Tutorial

Antoine MORVAN, Karol DESNOS

Open source tool : https://github.com/preesm/preesm
Web page: http://preesm.sourceforge.net/website/

Objectives

- Get familiar with PREESM
  - Play with SDF MoC/SLAM MoA/Y-codesign
  - Go from imperative to PREESM implementation
- Observe Results
- For Cerbero Partners:
  - grasp hints about how to integrate the tool in the global toolchain and apply the MoC/MoA approach on the use cases

! Objectives

- ~~Play with embedded systems/dev boards~~
- ~~Play with stereo vision or stabilization~~
- ~~See how to extend PREESM~~
  - ~~especially how to target a new processor~~
- ~~See communication implementation*~~
- ~~Go to production~~

Tutos available online

Outline (hidden slide)

- (Quick) Recall
- What we will do in this “hands on” session
- Sobel Filter
- Actor Decomposition
- Single Core
- Multi Core

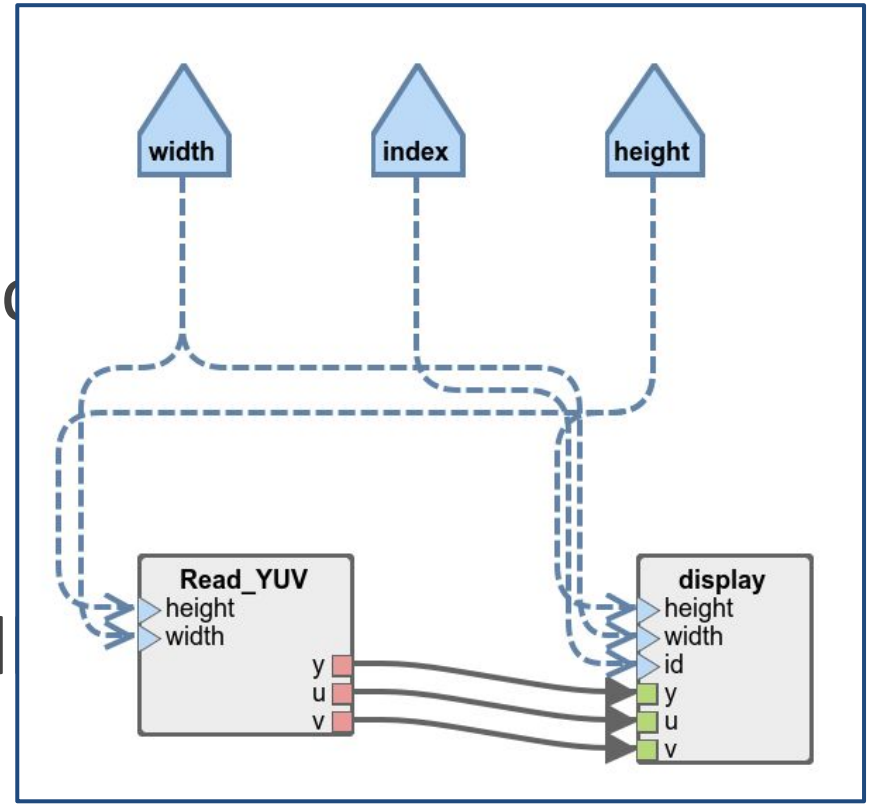
(Quick) Recall on Concepts

- Y- co design approach: keep archi and appli separate
- SDF: actors, fifo, hierarchy, delays, parameters
- SLAM: PE (processors), memories, links
- Scenario: “joint point”: mapping/scheduling/allocation directives

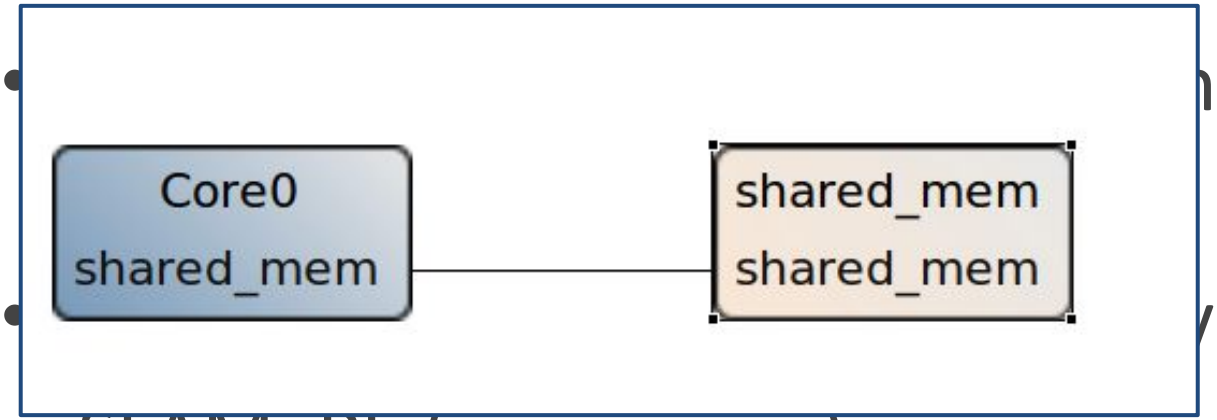
(Quick) Recall on Concepts

- Y- co design approach:
  separate
  mapping/scheduling/al

PiSDF (Parametrized Interfaced SDF)



(Quick) Recall on Concepts



- …ni and appli
- …ys, parameters
- SLAM: PE (processors), memories, links
- Scenario: "joi… …ves
  mapping/sch…

S-LAM Single Core
with shared memory
(simple Von Neuman)

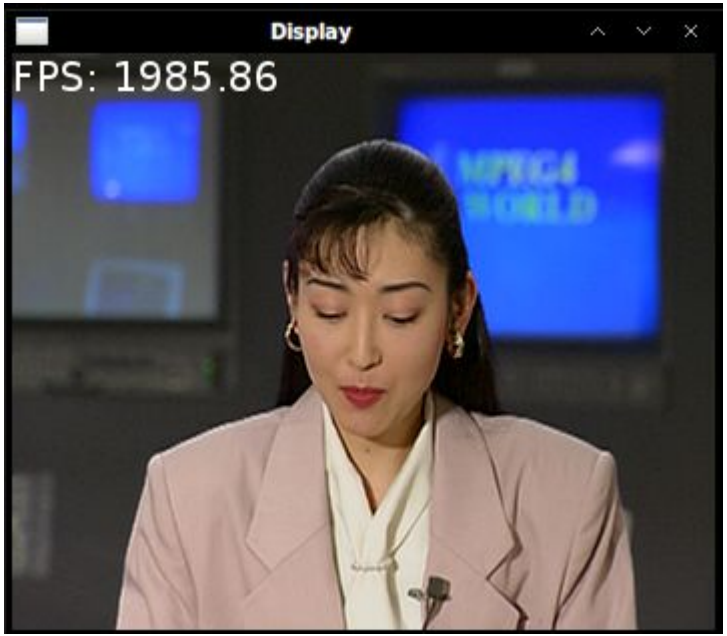
Practice Overview

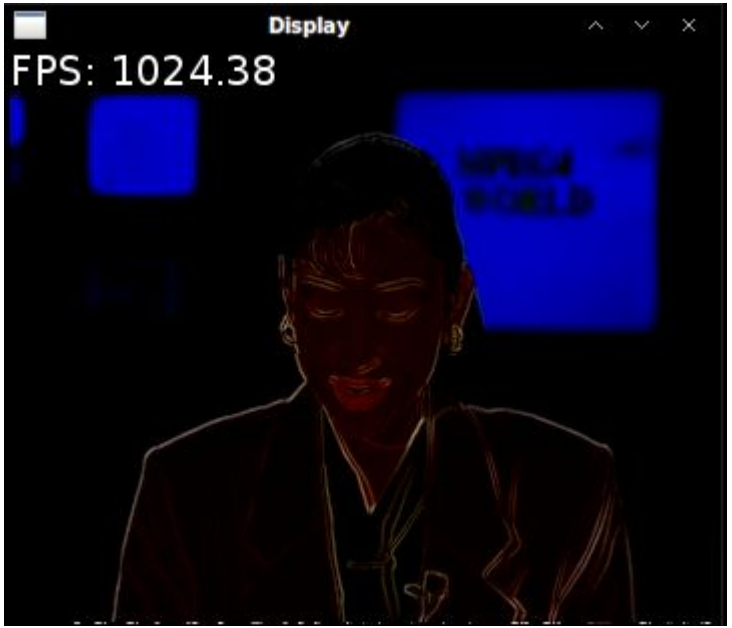
- Setup a simple example (Sobel filter), imperative + procedural MoC
- Design the same filter with actors in PREESM
- Generate code for 1 core archi, observe results
- Generate code for N cores archi, observe results

Materials @ http://preesm.sourceforge.net/website/data/uploads/tutorial_zips/preesm-tuto.zip

SourceForge is kinda down at the time of writing :-(

Sobel Edge Detection





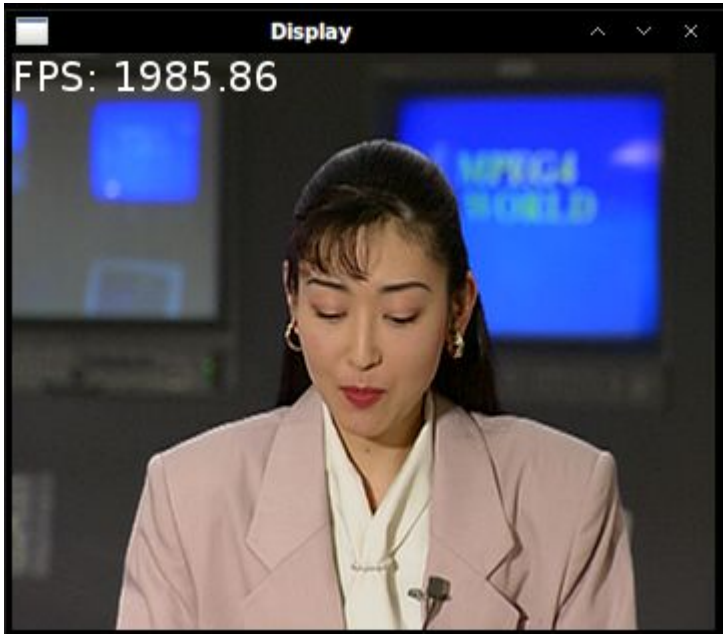
Display Init & Test

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define height 288

uchar y[101376];
uchar u[25344];
uchar v[25344];

void simple_display() {
    initReadYUV(width,height);
    yuvDisplayInit(0,width,height);
    while(!stopThreads){
        readYUV(width,height,y,u,v); // Read_YUV
        yuvDisplay(0,y,u,v); // display
    }
}
```



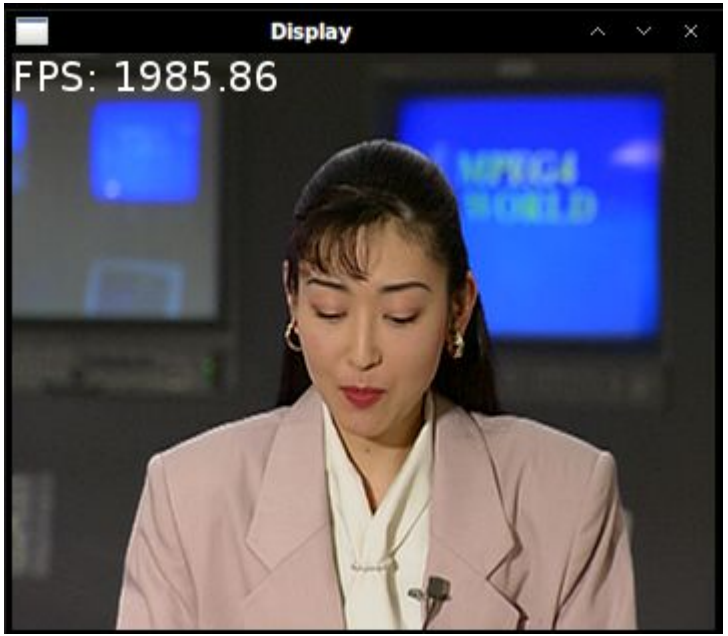
Display Init & Test

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define height 288

uchar y[101376];
uchar u[25344];
uchar v[25344];

void simple_display() {
   initReadYUV(width,height);
   yuvDisplayInit(0,width,height);
   while(!stopThreads){
      readYUV(width,height,y,u,v); // Read_YUV
      yuvDisplay(0,y,u,v); // display
   }
}
```



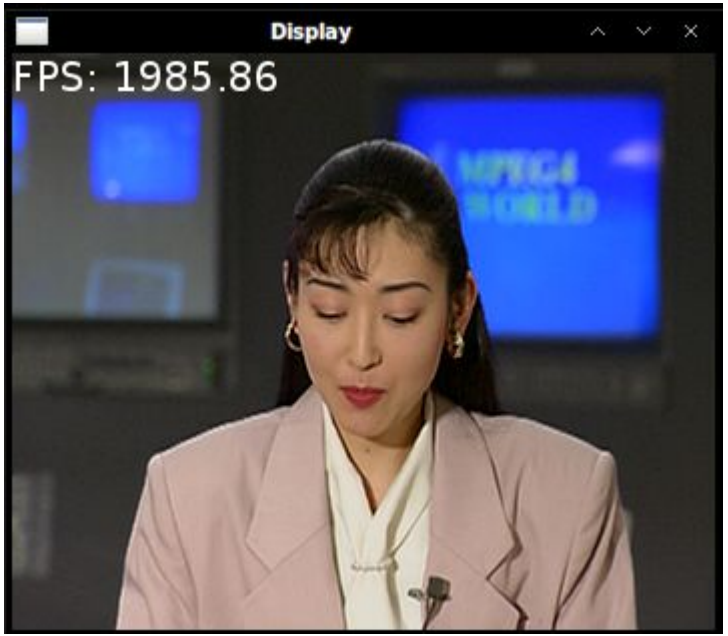
Display Init & Test

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define height 288

uchar y[101376];
uchar u[25344];
uchar v[25344];

void simple_display() {
    initReadYUV(width,height);
    yuvDisplayInit(0,width,height);
    while(!stopThreads){
        readYUV(width,height,y,u,v); // Read_YUV
        yuvDisplay(0,y,u,v); // display
    }
}
```



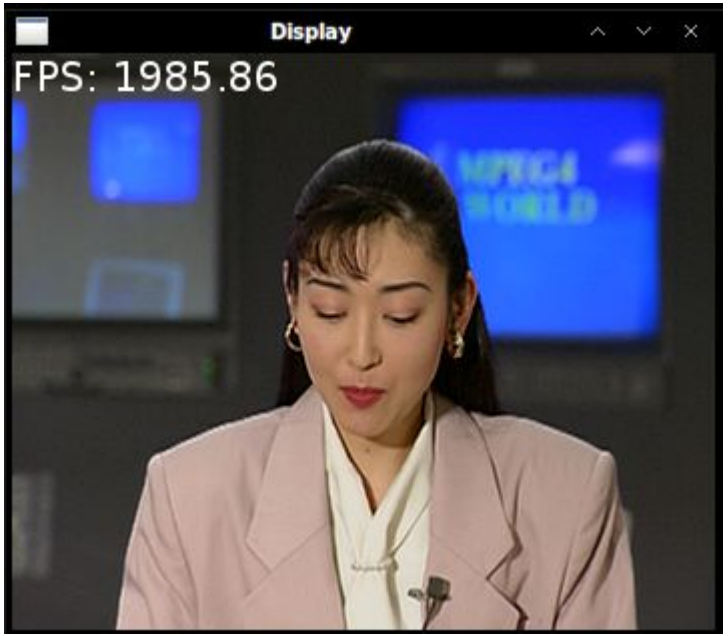
Display Init & Test

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define height 288

uchar y[101376];
uchar u[25344];
uchar v[25344];

void simple_display() {
  initReadYUV(width,height);
  yuvDisplayInit(0,width,height);
  while(!stopThreads){
    readYUV(width,height,y,u,v); // Read_YUV
    yuvDisplay(0,y,u,v); // display
  }
}
```



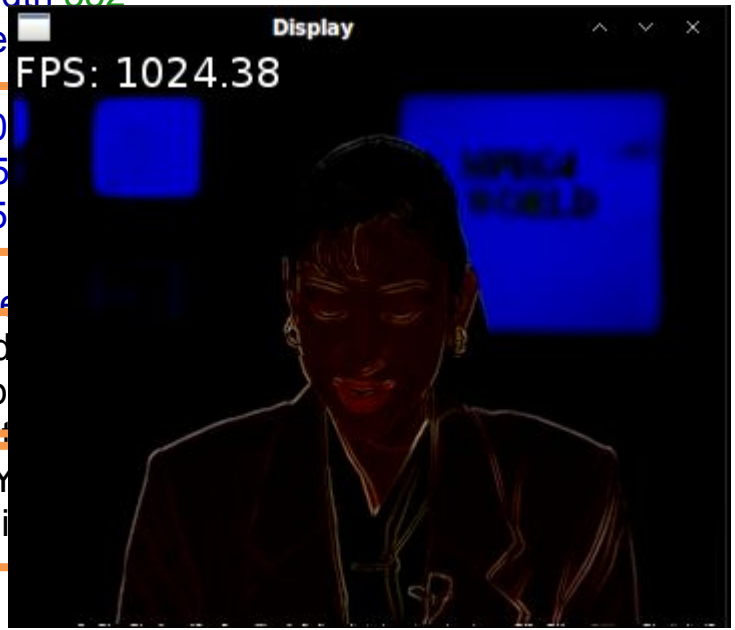
Insert Sobel Edge Detection

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define he

uchar y[10
uchar u[25
uchar v[25

void simple
    initRead
    yuvDisp
    while(!st
        readY
        yuvDi
    }
}
```




```
#define width 352
#define height 288

uchar y[101376]; uchar u[25344]; uchar v[25344];

void simple_sobel() {
    initReadYUV(width,height);
    yuvDisplayInit(0,width,height);
    //extra array
    uchar y_sobeled[101376];
    while(!stopThreads){
        readYUV(width,height,y,u,v); // Read_YUV
        sobel(width,height,y,y_sobeled); // Sobel
        yuvDisplay(0,y_sobeled,u,v); // display
    }
}
```

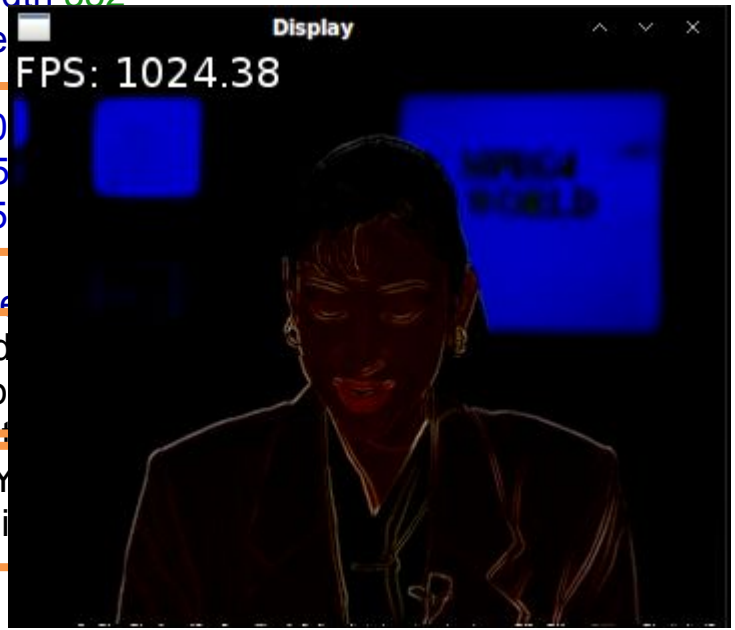
Insert Sobel Edge Detection

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define he

uchar y[10
uchar u[25
uchar v[25

void simple
    initRead
    yuvDisp
    while(!st
        readY
        yuvDi
    }
}
```




```
#define width 352
#define height 288

uchar y[101376]; uchar u[25344]; uchar v[25344];

void simple_sobel() {
    initReadYUV(width,height);
    yuvDisplayInit(0,width,height);
    //extra array
    uchar y_sobeled[101376];
    while(!stopThreads){
        readYUV(width,height,y,u,v); // Read_YUV
        sobel(width,height,y,y_sobeled); // Sobel
        yuvDisplay(0,y_sobeled,u,v); // display
    }
}
```

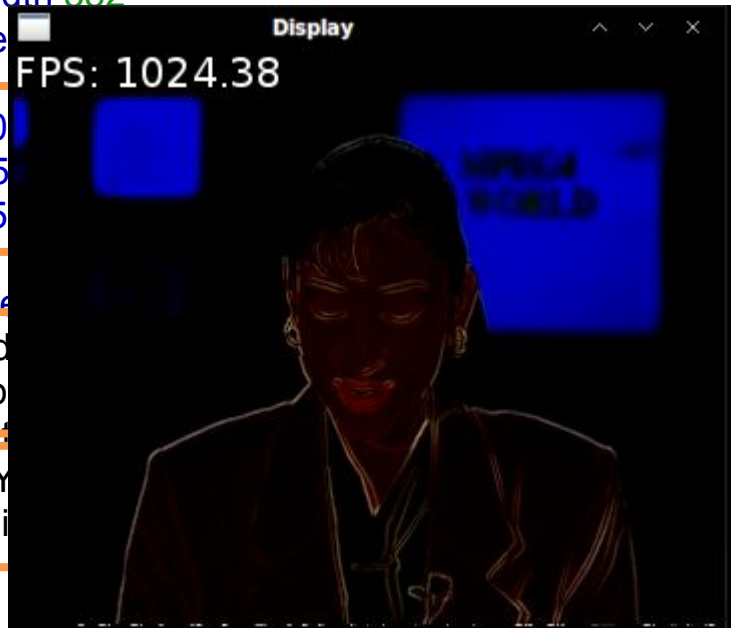
Insert Sobel Edge Detection

/org.ietr.preesm.sobel/Code/imperative-implem/sample_main.c

```
#define width 352
#define he

uchar y[10
uchar u[25
uchar v[25

void simple
  initRead
  yuvDisp
  while(!st
    readY
    yuvDi
  }
}
```




```
#define width 352
#define height 288

uchar y[101376]; uchar u[25344]; uchar v[25344];

void simple_sobel() {
  initReadYUV(width,height);
  yuvDisplayInit(0,width,height);
  //extra array
  uchar y_sobeled[101376];
  while(!stopThreads){
    readYUV(width,height,y,u,v); // Read_YUV
    sobel(width,height,y,y_sobeled); // Sobel
    yuvDisplay(0,y_sobeled,u,v); // display
  }
}
```

Try out

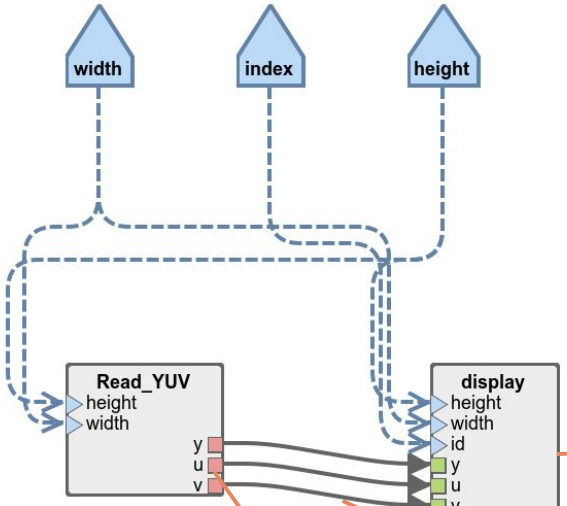
- Compile & Run Code
  - Open terminal in the Code folder

```
cd imperative-implem
make
./sobel
```

PREESMisation - Display

- Model Algorithm
- Model Architecture
- Join algo & archi
- Execute using Workflow

Actor Decomposition - Display



- (Index = 0; actor is used in stereo algo)
- Actor = init + body
- Size of data tokens
- Fifo token types

Name:	display
Refinement:	yuvDisplay.h
	loop: yuvDisplay(int id, unsigned char * y, unsigned char * u, unsigned char * v) init: yuvDisplayInit(int id, int width, int height)

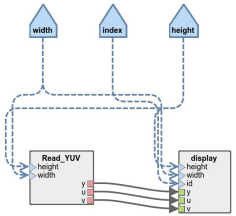
Name:	y
Expression:	height*width
Default Value:	101376

Data type:	uchar
Source port rate:	height/2*width/2
Default Value:	25344
Target port rate:	height/2*width/2
Default Value:	25344

/org.ietr.preesm.sobel/Algo/01-simple_display.diagram

PREESMisation - Display

- ~~Model Algorithm~~



- Model Architecture
- Join algo & archi
- Execute using Workflow

Single Core Target

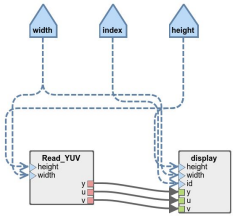
Property	Value
definition	x86
id	Core0
refinement	

Property	Value
definition	SHARED_MEM
id	shared_mem
refinement	
speed	1000000000

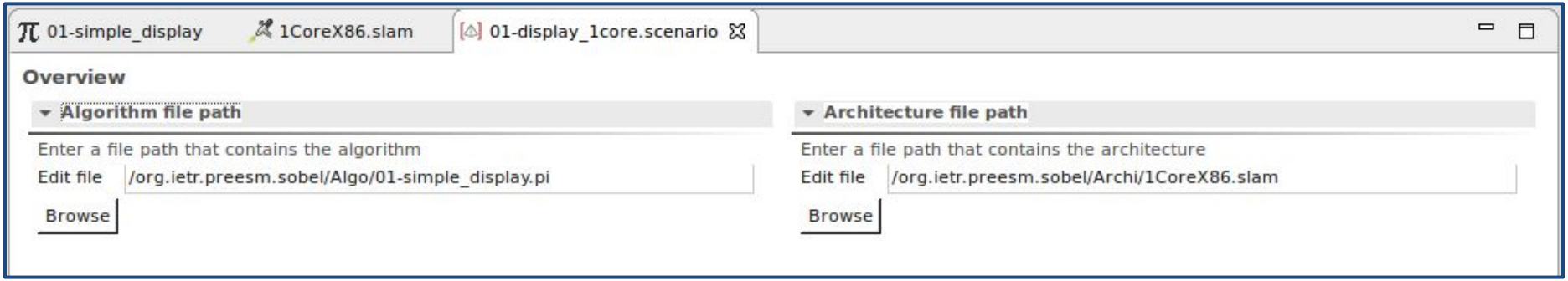
/org.ietr.preesm.sobel/Archi/1CoreX86.slam

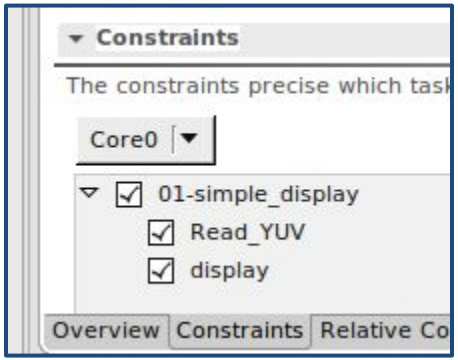
PREESMisation - Display

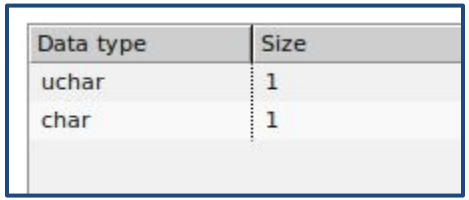
- ~~Model Algorithm~~
- ~~Model Architecture~~
- Join algo & archi
- Execute using Workflow



Join using Scenario







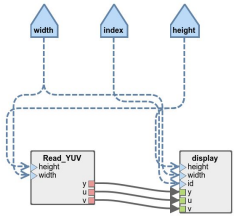
Data type	Size
uchar	1
char	1

- algo & archi join
- data type sizes
- target folders
- more accurate timings
- simulation parameters
- PE <-> actors affinity
- ...

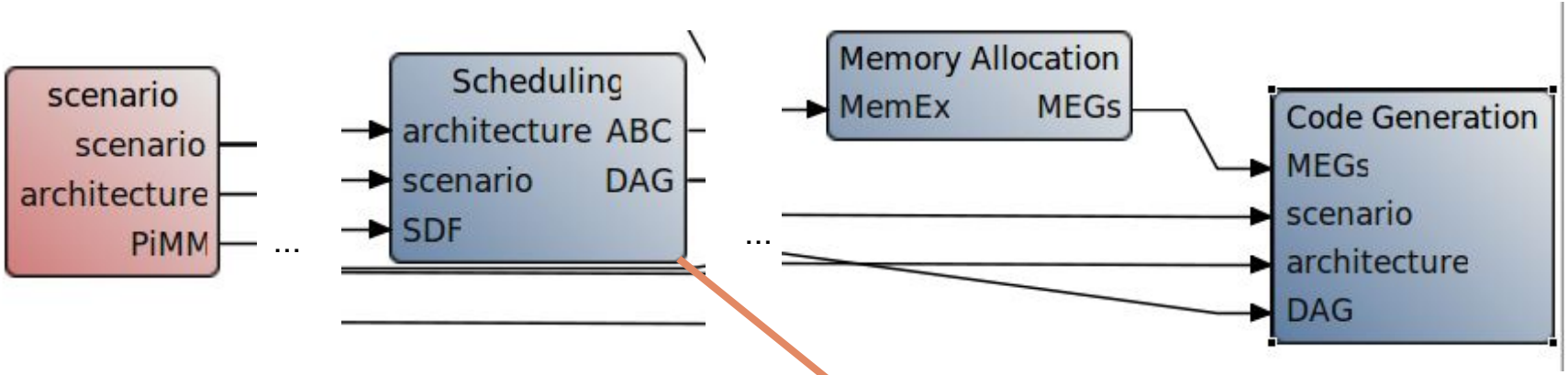
/org.ietr.preesm.sobel/Scenarios/01-display_1core.scenario

PREESMisation - Display

- ~~Model Algorithm~~
- ~~Model Architecture~~
- ~~Join algo & archi~~
- Execute using Workflow



Execution with a Workflow



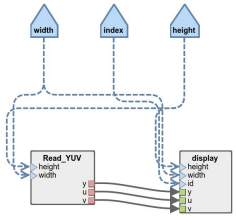
- Tells what to do
  - can reuse workflows
  - some Tasks are parametrized

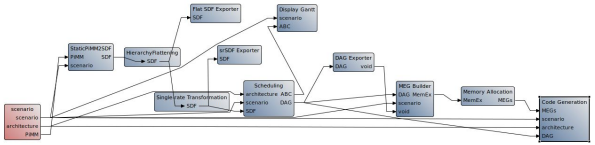
Name	Value
Check	true
balanceLoads	true
displaySolutions	true
edgeSchedType	Simple
fastLocalSearchTime	10
fastTime	100
iterationNr	0
iterationPeriod	0
listType	optimised
simulatorType	LooselyTimed

/org.ietr.preesm.sobel/Workflows/Codegen.workflow

PREESMisation - Display

- ~~Model Algorithm~~
- ~~Model Architecture~~
- ~~Join algo & archi~~
- ~~Execute using Workflow~~

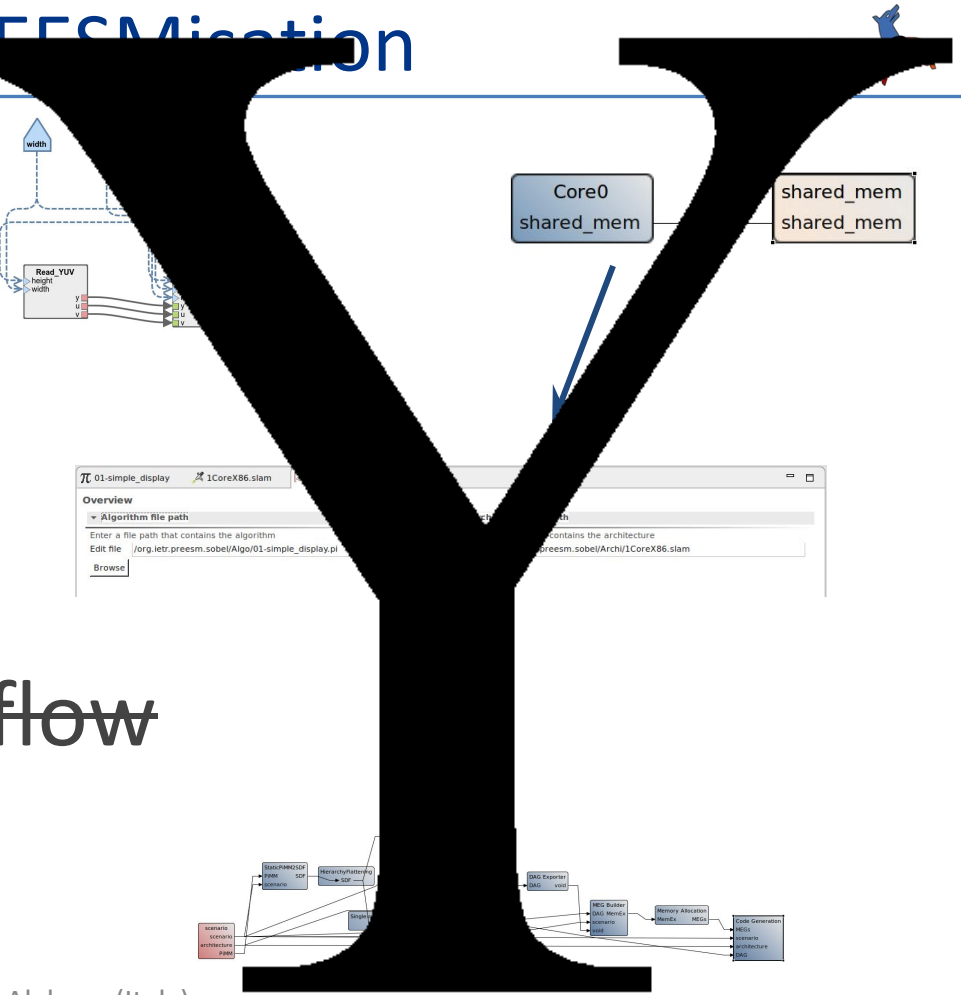




PREESMication

- ~~Model Algorithm~~
- ~~Model Architecture~~
- ~~Join algo & archi~~
- ~~Execute using Workflow~~

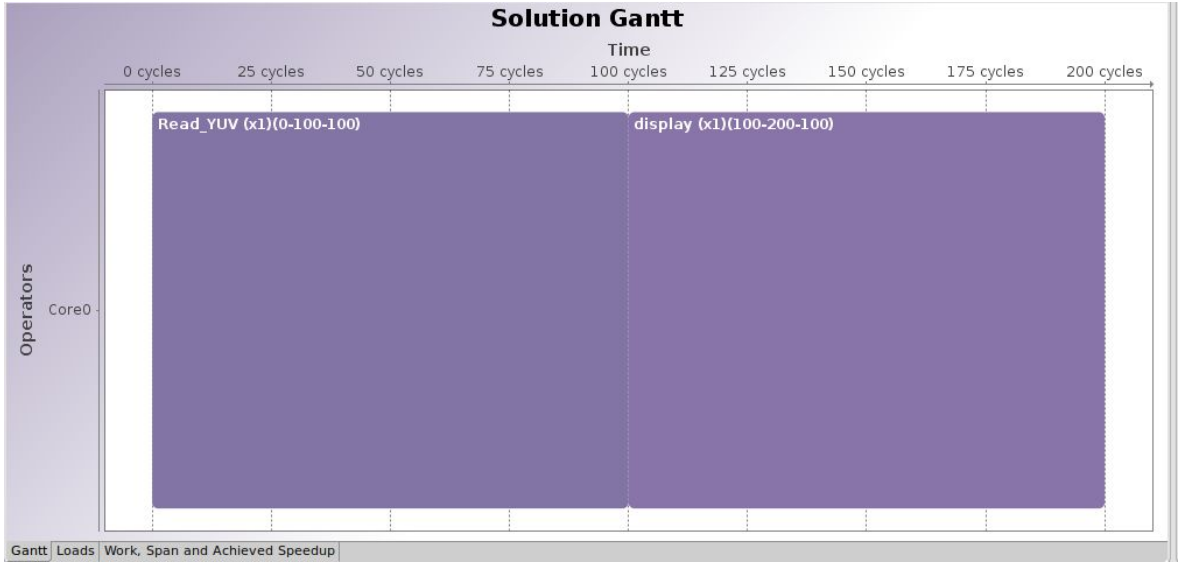
Y-co design flow



Try out

- Generate code
  - Right click on the Workflow / Preesm / Run Workflow
  - Select 01-display...scenario and click on OK
  - Observe ABC simulator results

Simulation Results



Operator	Load (%)	Memory (base unit)
Core0	100.0	152064

Try out

- Generate code
  - Right click on the Workflow / Preesm / Run Workflow
  - Select 01-display...scenario and click on OK
  - Observe ABC simulator results
  - **Observe generated code**
    - main.c
    - Core0.c

Generated Code - Display - main

Init threads & coms, launch, synchro, wait...

```
int main(void){
  pthread_barrier_init(&iter_barrier, NULL, 1);
  communicationInit();
  // ...
  pthread_create(&threadCore0, NULL, computationThread_Core0, NULL);
  // Waiting for thread terminations
  pthread_join(threadCore0,NULL);
}
```

Generated Code - Display - Core0


```
char Shared[152064]; //  size:= 152064*char
uchar *const u__u__0 = (uchar*) (Shared+0);  // Read_YUV_u > display_u size:= 25344*uchar
uchar *const v__v__0 = (uchar*) (Shared+25344);  // Read_YUV_v > display_v size:= 25344*uchar
//...
void *computationThread_Core0(void *arg){
    // Initialisation(s)
    initReadYUV(352/*width*/,288/*height*/); // Read_YUV
    yuvDisplayInit(0/*id*/,352/*width*/,288/*height*/); // display
    // Begin the execution loop
   //...
    while(1){
        pthread_barrier_wait(&iter_barrier);
        readYUV(352/*width*/,288/*height*/,y__y__0,u__u__0,v__v__0); // Read_YUV
        yuvDisplay(0/*id*/,y__y__0,u__u__0,v__v__0); // display
    }
}
```

Generated Code - Display - Core0

```
char Shared[152064]; //  size:= 152064*char
uchar *const u__u__0 = (uchar*) (Shared+0);  // Read_YUV_u > displa
uchar *const v__v__0 = (uchar*) (Shared+25344);  // Read_YUV_v > d
//...
void *computationThread_Core0(void *arg){
    // Initialisation(s)
    initReadYUV(352/*width*/,288/*height*/); // Read_YUV
    yuvDisplayInit(0/*id*/,352/*width*/,288/*height*/); // display
    // Begin the execution loop
   //...
    while(1){
        pthread_barrier_wait(&iter_barrier);
        readYUV(352/*width*/,288/*height*/,y__y__0,u__u__0,v__v__0); // Read_YUV
        yuvDisplay(0/*id*/,y__y__0,u__u__0,v__v__0); // display
    }
}
```

- Total memory requirements
- Custom allocation

Generated Code - Display - Core0

```
char Shared[152064]; //  size:= 152064*char
uchar *const u__u__0 = (uchar*) (Shared+0);  // Read_YUV_u > displa
uchar *const v__v__0 = (uchar*) (Shared+25344);  // Read_YUV_v > d
//...
void *computationThread_Core0(void *arg){
  // Initialisation(s)
  initReadYUV(352/*width*/,288/*height*/); // Read_YUV
  yuvDisplayInit(0/*id*/,352/*width*/,288/*height*/); // display
  // Begin the execution loop
  //...
  while(1){
    pthread_barrier_wait(&iter_barrier);
    readYUV(352/*width*/,288/*height*/,y__y__0,u__u__0,v__v__0); // Read_YUV
    yuvDisplay(0/*id*/,y__y__0,u__u__0,v__v__0); // display
  }
}
```

- Generate calls for init & loop
- Automatically replaces parameter values
- Properly pass memory addresses as arguments

Try out

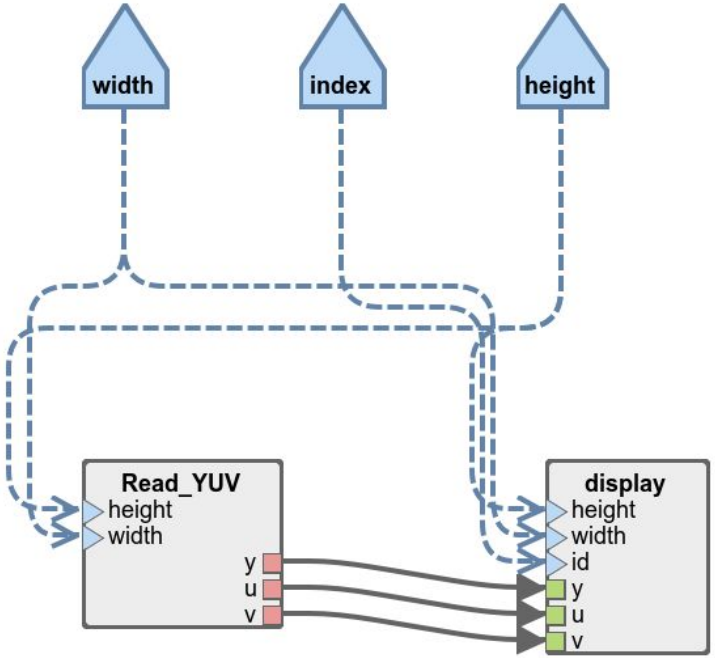
- Compile & Run Code
  - Open terminal in the Code folder

```
sh CMakeGCC.sh
cd bin/make && make
./Release/sobel
```

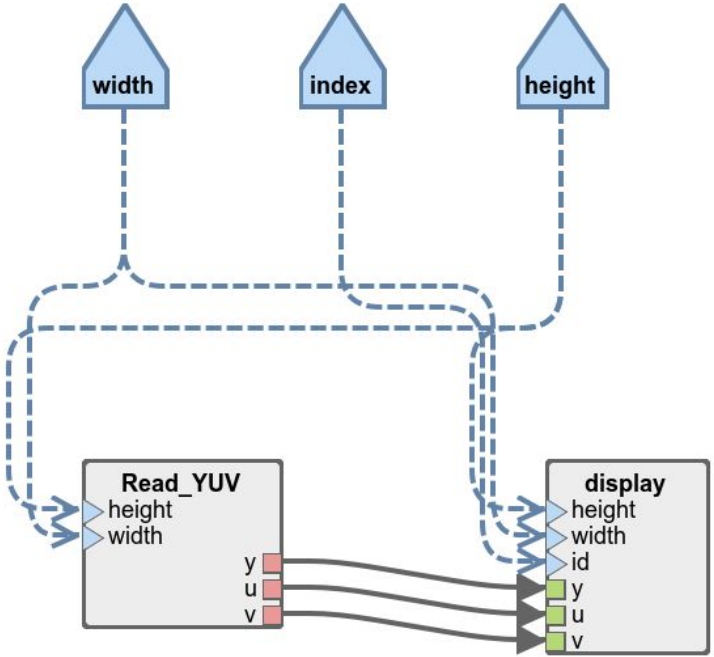
Insert Sobel Filter

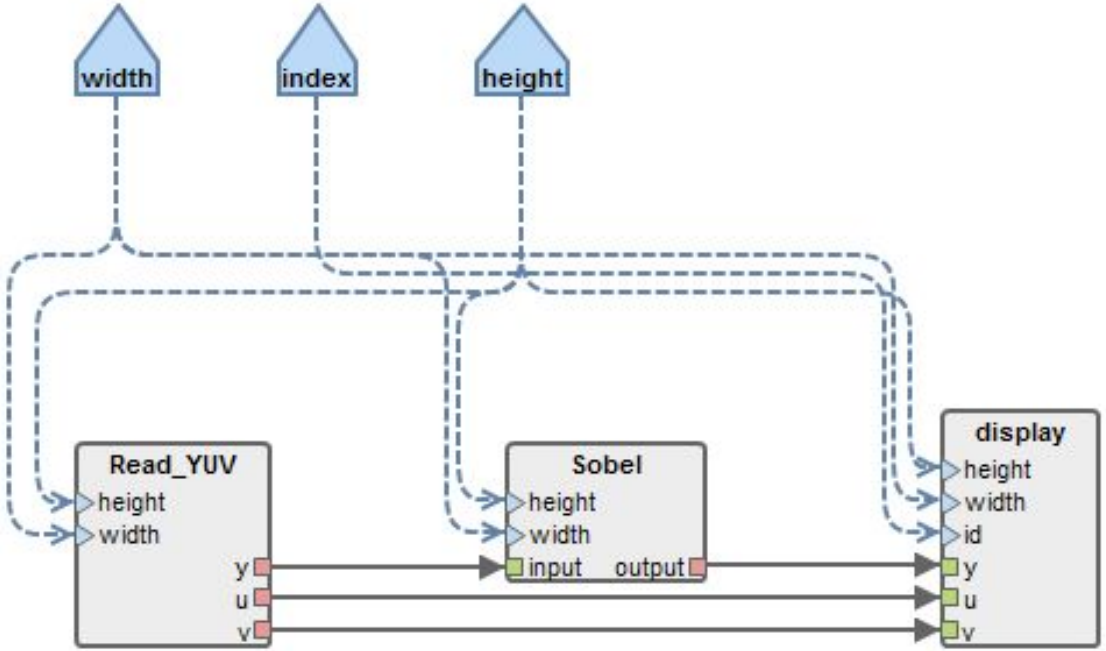
- Only edit algorithm
  - well, also scenario, but not architecture
- Insert sobel actor
- Run workflow (generate code)
- Observe Gantt & Code
- Try out

Insert Sobel Filter

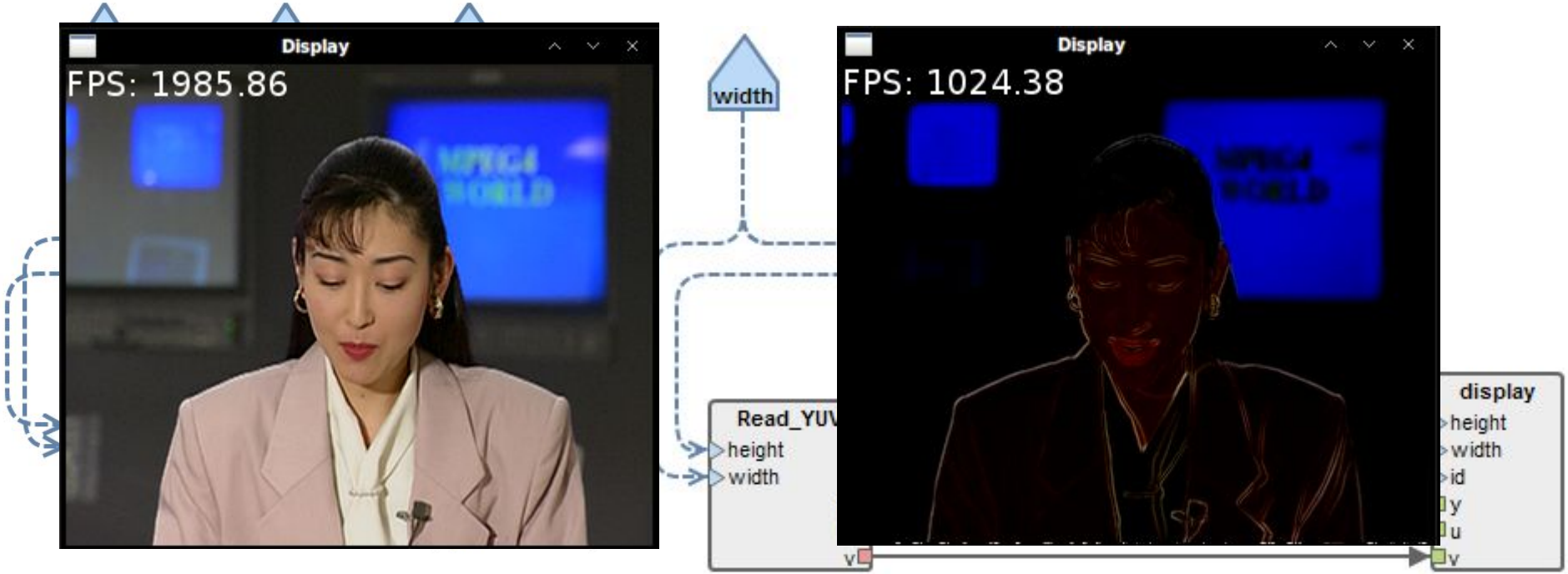


Insert Sobel Filter

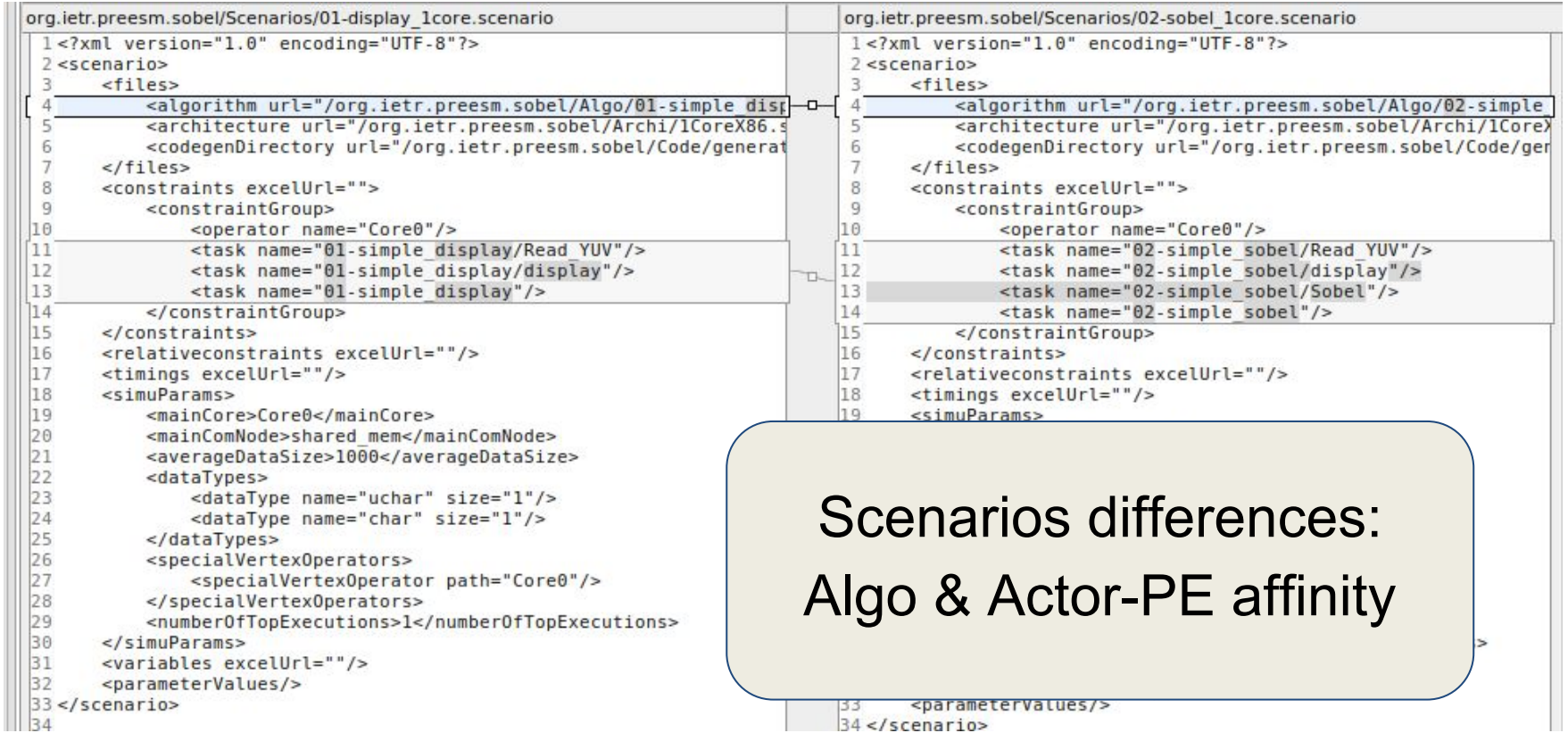




Insert Sobel Filter



Insert Sobel Filter



org.ietr.preesm.sobel/Scenarios/01-display_1core.scenario

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
    <files>
        <algorithm url="/org.ietr.preesm.sobel/Algo/01-simple_disp
        <architecture url="/org.ietr.preesm.sobel/Archi/1CoreX86.s
        <codegenDirectory url="/org.ietr.preesm.sobel/Code/generat
    </files>
    <constraints excelUrl="">
        <constraintGroup>
            <operator name="Core0"/>
            <task name="01-simple_display/Read_YUV"/>
            <task name="01-simple_display/display"/>
            <task name="01-simple_display"/>
        </constraintGroup>
    </constraints>
    <relativeconstraints excelUrl=""/>
    <timings excelUrl=""/>
    <simuParams>
        <mainCore>Core0</mainCore>
        <mainComNode>shared_mem</mainComNode>
        <averageDataSize>1000</averageDataSize>
        <dataTypes>
            <dataType name="uchar" size="1"/>
            <dataType name="char" size="1"/>
        </dataTypes>
        <specialVertexOperators>
            <specialVertexOperator path="Core0"/>
        </specialVertexOperators>
        <numberOfTopExecutions>1</numberOfTopExecutions>
    </simuParams>
    <variables excelUrl=""/>
    <parameterValues/>
</scenario>
```

org.ietr.preesm.sobel/Scenarios/02-sobel_1core.scenario

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
    <files>
        <algorithm url="/org.ietr.preesm.sobel/Algo/02-simple_
        <architecture url="/org.ietr.preesm.sobel/Archi/1CoreX
        <codegenDirectory url="/org.ietr.preesm.sobel/Code/ger
    </files>
    <constraints excelUrl="">
        <constraintGroup>
            <operator name="Core0"/>
            <task name="02-simple_sobel/Read_YUV"/>
            <task name="02-simple_sobel/display"/>
            <task name="02-simple_sobel/Sobel"/>
            <task name="02-simple_sobel"/>
        </constraintGroup>
    </constraints>
    <relativeconstraints excelUrl=""/>
    <timings excelUrl=""/>
    <simuParams>
    <parameterValues/>
</scenario>
```

Scenarios differences:
Algo & Actor-PE affinity

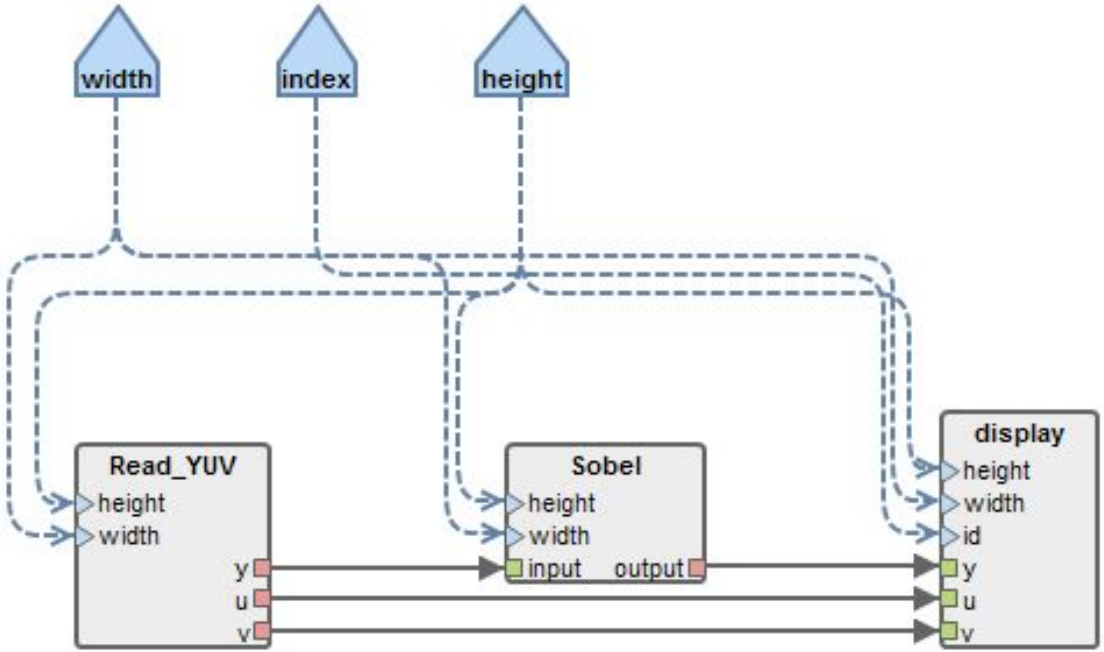
Try out

- Compile & Run Code
  - Open terminal in the Code folder

```
sh CMakeGCC.sh
cd bin/make && make
./Release/sobel
```

Multi Core Target

Is there any parallelism ?



Multi Core Target

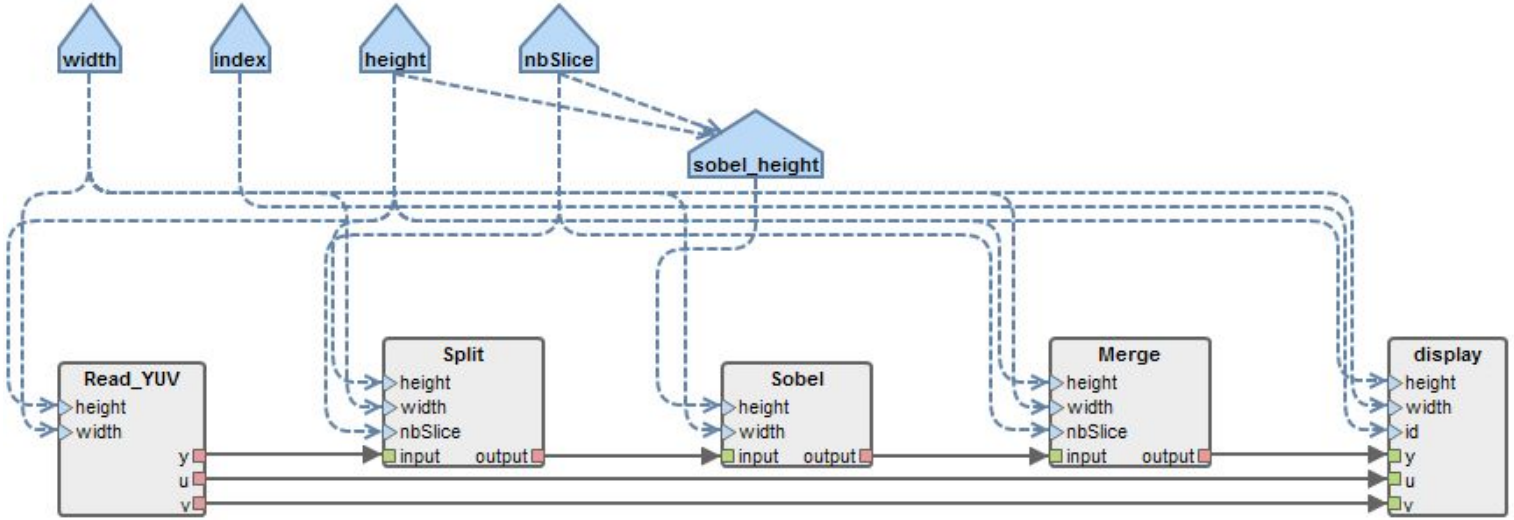
Is there any parallelism ?

```
while(1){
    pthread_barrier_wait(&iter_barrier);
```

NO Pipeline

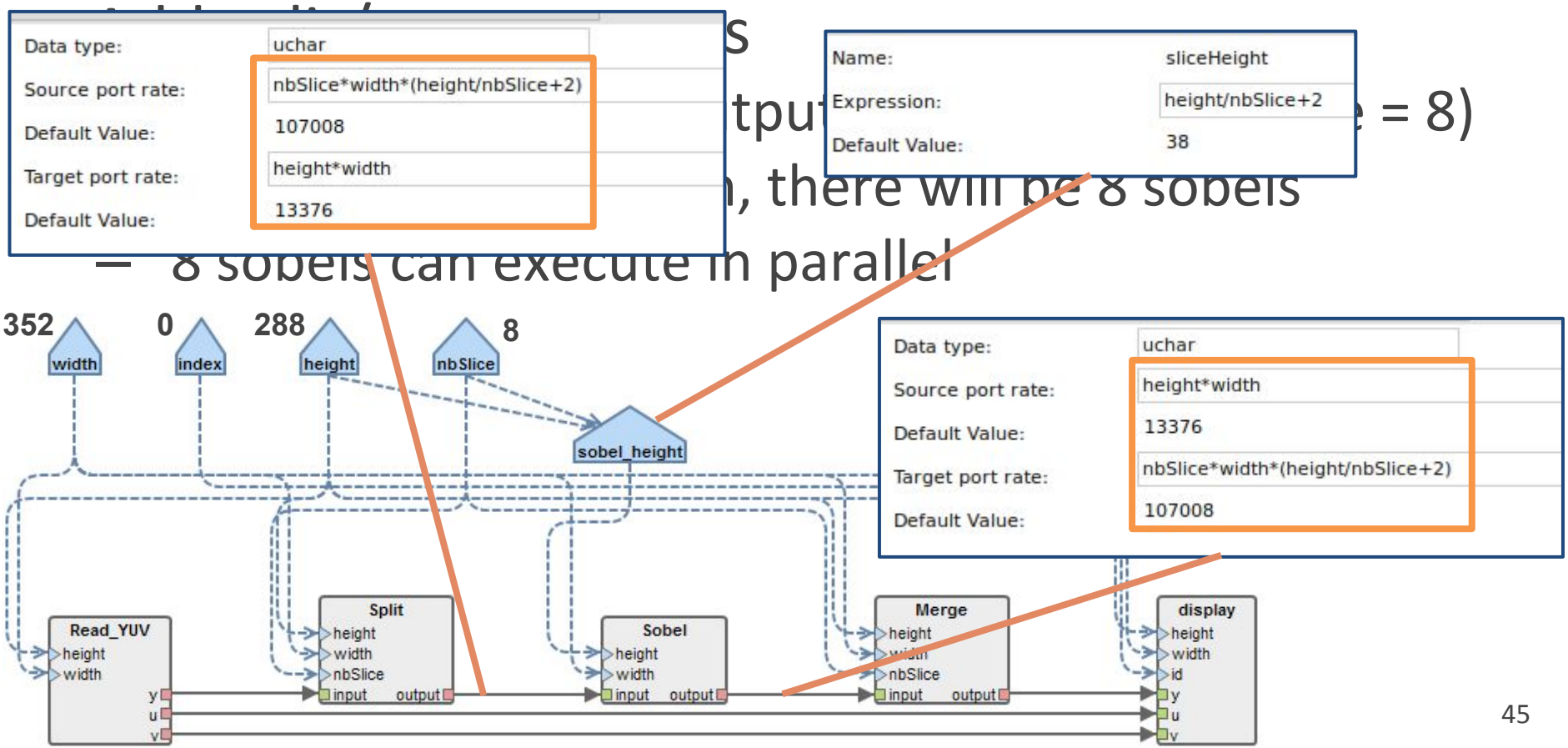
Multi Core Target

- Add split/merge actors
  - sobel_input =split_output/nbSlice (i.e. nbSlice = 8)
  - for one split execution, there will be 8 sobels
  - 8 sobels can execute in parallel

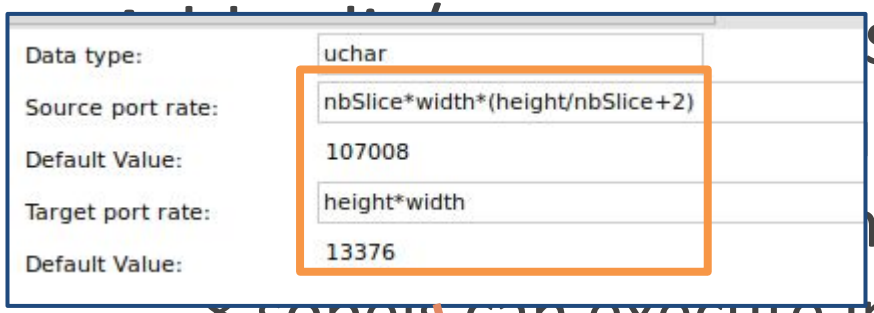


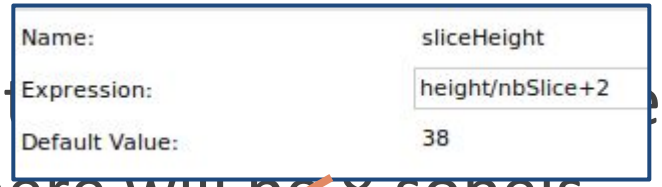
Multi Core Target

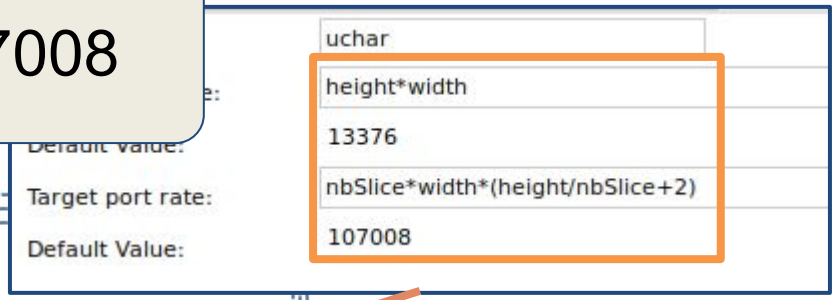
- S
- tput = 8)
- , there will be 8 sobels
- 8 sobels can execute in parallel



Multi Core Target

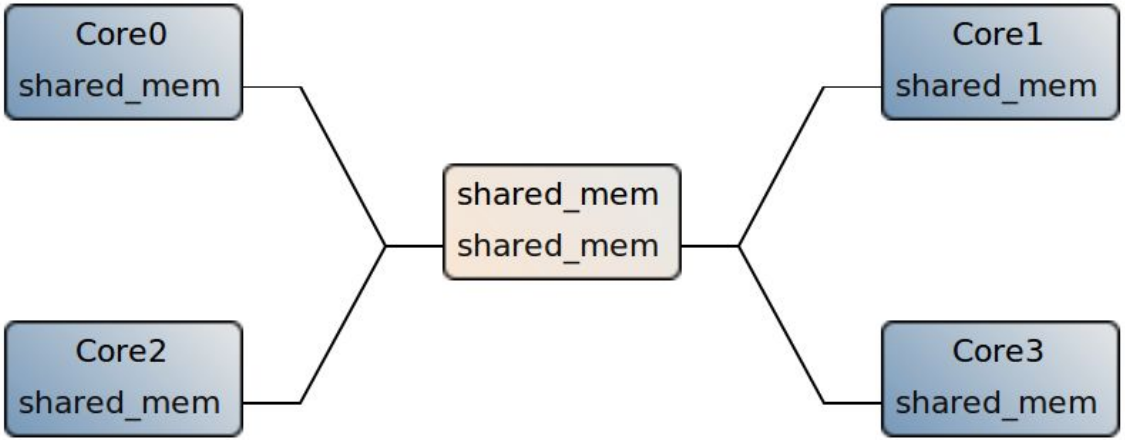




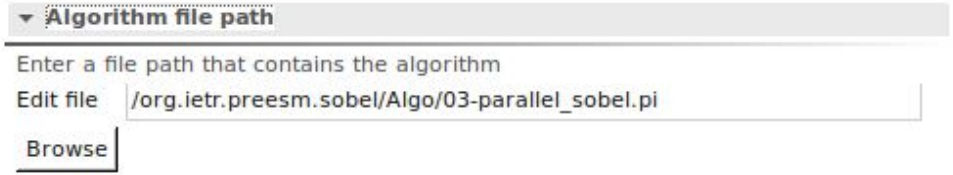


13376 * 8 = 107008

Multi Core Target

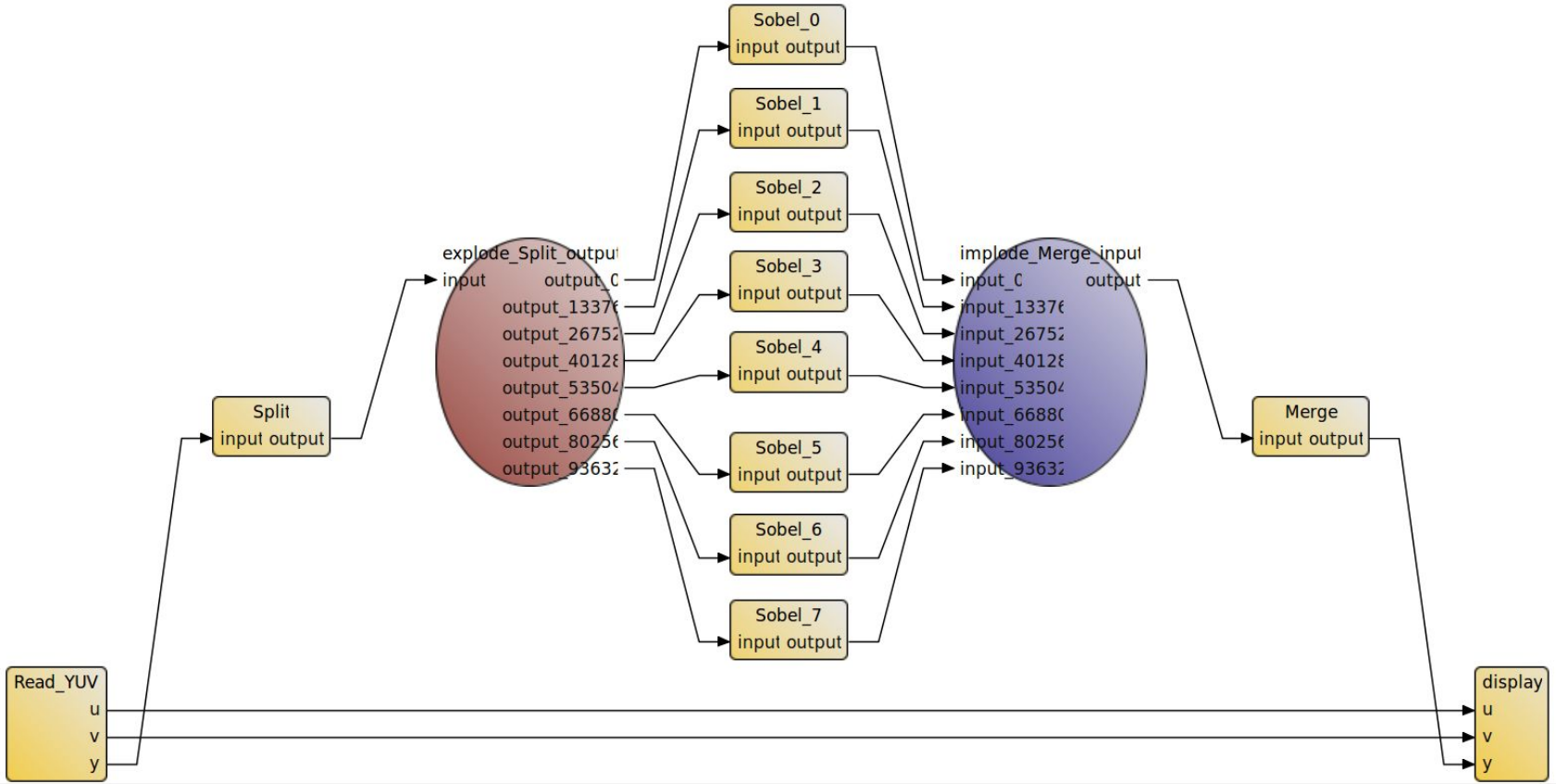


- Change architecture
- Change scenario
- Run workflow
  - (generate code)



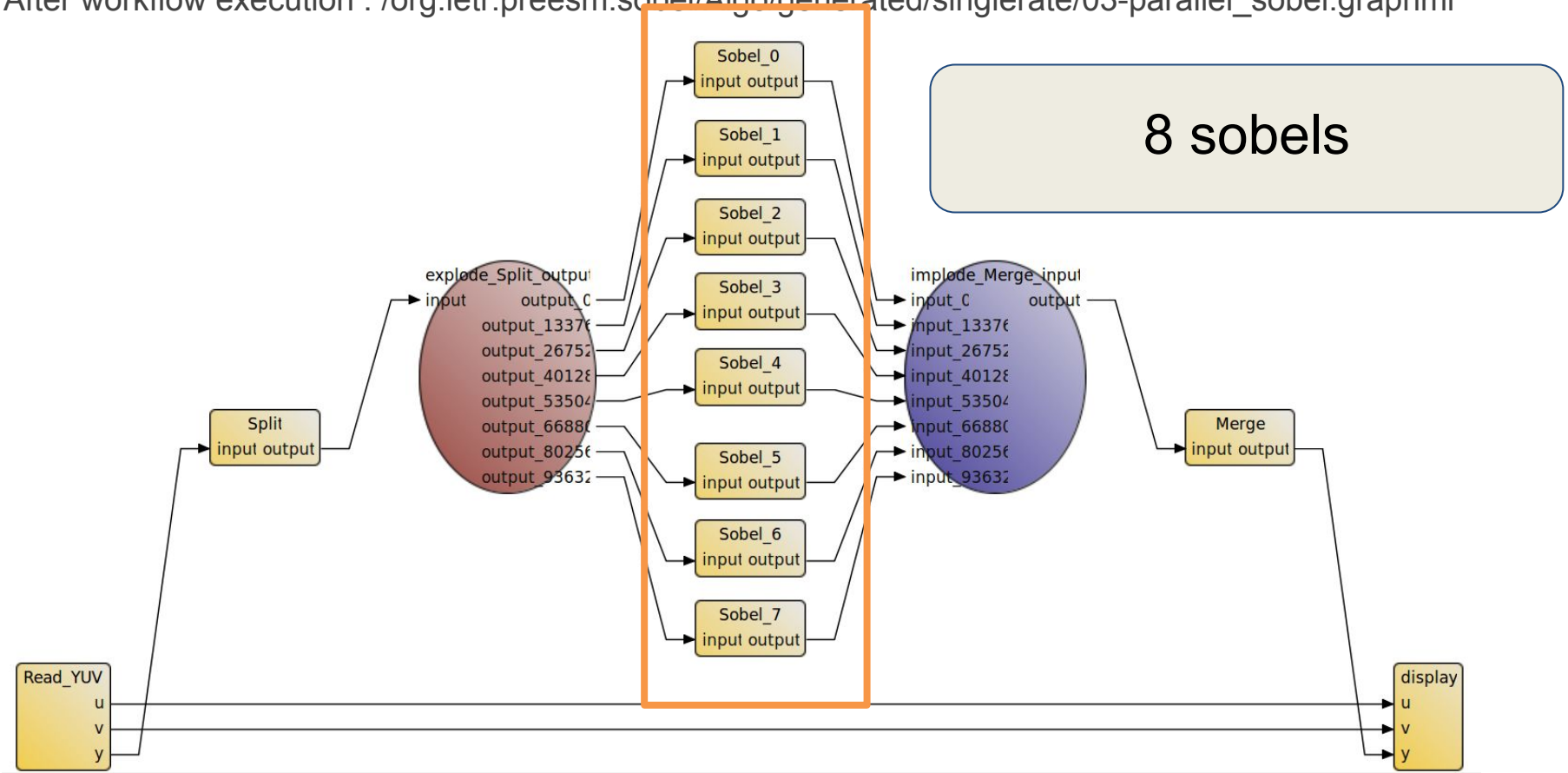
Generated Single Rate DAG

After workflow execution : /org.ietr.preesm.sobel/Algo/generated/singlerate/03-parallel_sobel.graphml

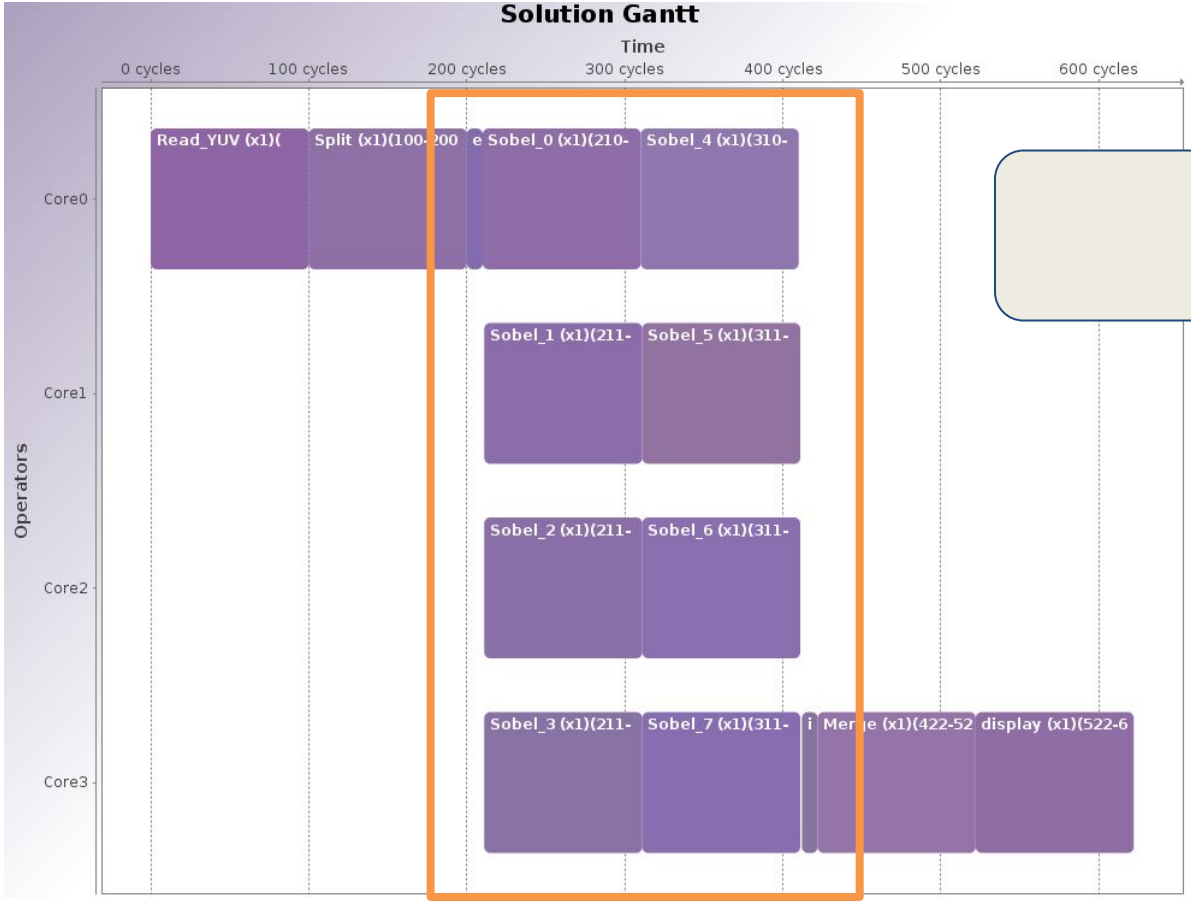


Generated Single Rate DAG

After workflow execution : /org.ietr.preesm.sobel/Algo/generated/singlerate/03-parallel_sobel.graphml



Observe Simulation



8 sobels

Operator	Load (%)
Core0	65.92
Core1	32.16
Core2	32.16
Core3	65.92

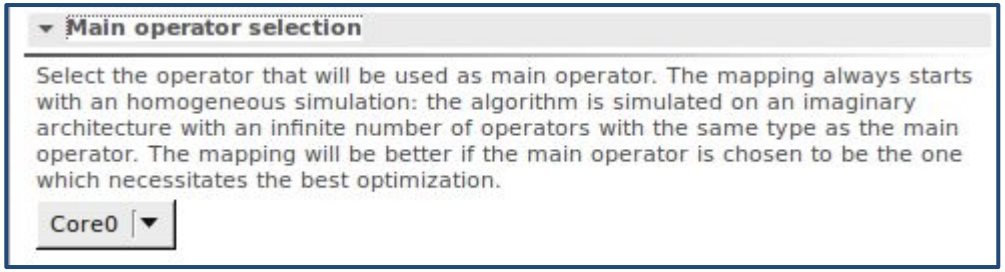
Generated Code - Sobel - main

```
//main()...
//…
pthread_create(&threadCore0, NULL, computationThread_Core0, NULL);
pthread_create(&threadCore1, NULL, computationThread_Core1, NULL);
pthread_create(&threadCore2, NULL, computationThread_Core2, NULL);
pthread_create(&threadCore3, NULL, computationThread_Core3, NULL);
//...
```

Launch more threads.

Generated Code - Sobel - Core0

Core0 = main operator



Generated Code - Sobel - Core0

Core0 = main operator

- in charge of shared memory (because main op);
- in charge of read & split (see Gantt chart);
- in charge of 2 sobels (0 and 4, see Gantt chart);

```
char Shared[681472]; //  size:= 681472*char
char *const Read_YUV__display__0 = (char*) (Shared+0);  // Read_YUV > display size:= 50688*char
char *const Read_YUV__Split__0 = (char*) (Shared+50688);  // Read_YUV > Split size:= 101376*char
char *const Split__explode_Split_output__0 = (char*) (Shared+152064);  // Split > explode_Split_output size:= 107008*char
char *const explode_Split_output__Sobel__0 = (char*) (Shared+259072);  // explode_Split_output > Sobel_0 size:= 13376*ch
char *const explode_Split_output__Sobel__1 = (char*) (Shared+272448);  // explode_Split_output > Sobel_1 size:= 13376*ch
char *const explode_Split_output__Sobel__2 = (char*) (Shared+285824);  // explode_Split_output > Sobel_2 size:= 13376*ch
char *const explode_Split_output__Sobel__3 = (char*) (Shared+299200);  // explode_Split_output > Sobel_3 size:= 13376*ch
char *const explode_Split_output__Sobel__4 = (char*) (Shared+312576);  // explode_Split_output > Sobel_4 size:= 13376*ch
char *const explode_Split_output__Sobel__5 = (char*) (Shared+325952);  // explode_Split_output > Sobel_5 size:= 13376*ch
char *const explode_Split_output__Sobel__6 = (char*) (Shared+339328);  // explode_Split_output > Sobel_6 size:= 13376*ch
char *const explode_Split_output__Sobel__7 = (char*) (Shared+352704);  // explode_Split_output > Sobel_7 size:= 13376*ch
char *const Sobel_0__implode_Merge_input__0 = (char*) (Shared+366080);  // Sobel_0 > implode_Merge_input size:= 13376*cha
char *const Sobel_4__implode_Merge_input__0 = (char*) (Shared+379456);  // Sobel_4 > implode_Merge_input size:= 13376*cha
char *const Sobel_1__implode_Merge_input__0 = (char*) (Shared+392832);  // Sobel_1 > implode_Merge_input size:= 13376*cha
char *const Sobel_2__implode_Merge_input__0 = (char*) (Shared+406208);  // Sobel_2 > implode_Merge_input size:= 13376*cha
char *const Sobel_3__implode_Merge_input__0 = (char*) (Shared+419584);  // Sobel_3 > implode_Merge_input size:= 13376*cha
char *const Sobel_5__implode_Merge_input__0 = (char*) (Shared+432960);  // Sobel_5 > implode_Merge_input size:= 13376*cha
char *const Sobel_6__implode_Merge_input__0 = (char*) (Shared+446336);
char *const Sobel_7__implode_Merge_input__0 = (char*) (Shared+45
char *const implode_Merge_input__Merge__0 = (char*) (Shared+473
char *const Merge__display__0 = (char*) (Shared+580096);  // Me
uchar *const u__u__0 = (uchar*) (Shared+0);  // Read_YUV_u > di
uchar *const v__v__0 = (uchar*) (Shared+25344);  // Read_YUV_v
uchar *const y__input__0 = (uchar*) (Shared+50688);  // Read_YU
uchar *const output__input__0 = (uchar*) (Shared+152064);  // S
uchar *const output_0__input__0 = (uchar*) (Shared+259072);  //
uchar *const output_13376__input__0 = (uchar*) (Shared+272448);
uchar *const output_26752__input__0 = (uchar*) (Shared+285824);
uchar *const output_40128__input__0 = (uchar*) (Shared+299200);
uchar *const output_53504__input__0 = (uchar*) (Shared+312576);
uchar *const output_66880__input__0 = (uchar*) (Shared+325952);
uchar *const output_80256__input__0 = (uchar*) (Shared+339328);
uchar *const output_93632__input__0 = (uchar*) (Shared+352704);  // explode_Split_output_output_93632 > Sobel_7_input si
uchar *const output__input_0__0 = (uchar*) (Shared+366080);  // Sobel_0_output > implode_Merge_input_input_0 size:= 1337
uchar *const output__input_53504__0 = (uchar*) (Shared+379456);  // Sobel_4_output > implode_Merge_input_input_53504 size
uchar *const output__input_13376__0 = (uchar*) (Shared+392832);  // Sobel_1_output > implode_Merge_input_input_13376 size
uchar *const output__input_26752__0 = (uchar*) (Shared+406208);  // Sobel_2_output > implode_Merge_input_input_26752 size
uchar *const output__input_40128__0 = (uchar*) (Shared+419584);  // Sobel_3_output > implode_Merge_input_input_40128 size
uchar *const output__input_66880__0 = (uchar*) (Shared+432960);  // Sobel_5_output > implode_Merge_input_input_66880 size
uchar *const output__input_80256__0 = (uchar*) (Shared+446336);  // Sobel_6_output > implode_Merge_input_input_80256 size
uchar *const output__input_93632__0 = (uchar*) (Shared+459712);  // Sobel_7_output > implode_Merge_input_input_93632 size
uchar *const output__input__1 = (uchar*) (Shared+473088);  // implode_Merge_input_output > Merge_input size:= 107008*uchar
uchar *const output__y__0 = (uchar*) (Shared+580096);  // Merge_output > display_y size:= 101376*uchar
```

C

-
-
-

);

Allocation of memory
+ declaration of buffer zones within the memory

Generated Code - Sobel - Core0

```
split(8/*nbSlice*/,352/*width*/,288/*height*/,y__input__0,output__input__0); // Split
// Fork explode_Split_output
{
    memcpy(output_0__input__0+0, output__input__0+0, 13376*sizeof(uchar));
    memcpy(output_13376__input__0+0, output__input__0+13376, 13376*sizeof(uchar));
    memcpy(output_26752__input__0+0, output__input__0+26752, 13376*sizeof(uchar));
    memcpy(output_40128__input__0+0, output__input__0+40128, 13376*sizeof(uchar));
    memcpy(output_53504__input__0+0, output__input__0+53504, 13376*sizeof(uchar));
    memcpy(output_66880__input__0+0, output__input__0+66880, 13376*sizeof(uchar));
    memcpy(output_80256__input__0+0, output__input__0+80256, 13376*sizeof(uchar));
    memcpy(output_93632__input__0+0, output__input__0+93632, 13376*sizeof(uchar));
}
sendStart(0, 1); // Core0 > Core1: explode_Split_output__Sobel__1
sendEnd(); // Core0 > Core1: explode_Split_output__Sobel__1
sendStart(0, 2); // Core0 > Core2: explode_Split_output__Sobel__2
sendEnd(); // Core0 > Core2: explode_Split_output__Sobel__2
sendStart(0, 3); // Core0 > Core3: explode_Split_output__Sobel__3
sendEnd(); // Core0 > Core3: explode_Split_output__Sobel__3
sendStart(0, 1); // Core0 > Core1: explode_Split_output__Sobel__5
sendEnd(); // Core0 > Core1: explode_Split_output__Sobel__5
sendStart(0, 2); // Core0 > Core2: explode_Split_output__Sobel__6
sendEnd(); // Core0 > Core2: explode_Split_output__Sobel__6
sendStart(0, 3); // Core0 > Core3: explode_Split_output__Sobel__7
sendEnd(); // Core0 > Core3: explode_Split_output__Sobel__7
```

:ause main op);

t);

Split + send to all cores

Does the actual copy

see /org.ietr.preesm.sobel/Code/src/communication.c

Generated Code - Sobel - Core0

Core0 = main operator

- in char ause main op);
- in char);
- in char e Gantt chart);

Compute sobel*2 +
Send results

```
sobel(352/*width*/,38/*height*/,output_0__input__0,output__input_0__0); // Sobel_0
sendStart(0, 3); // Core0 > Core3: Sobel_0__implode_Merge_input__0
sendEnd(); // Core0 > Core3: Sobel_0__implode_Merge_input__0
sobel(352/*width*/,38/*height*/,output_53504__input__0,output__input_53504__0); // Sobel_4
sendStart(0, 3); // Core0 > Core3: Sobel_4__implode_Merge_input__0
sendEnd(); // Core0 > Core3: Sobel_4__implode_Merge_input__0
```

Generated Code - Sobel - Core3

- access to shared memory;
- in charge of receives + 2 sobels;
- in charge of merge & display (see Gantt chart);

Generated Code - Sobel - Core3

```
// Core Global Declaration
extern pthread_barrier_t iter_barrier;
extern int stopThreads;
extern char *const Read_YUV__display__0;  // Read_YUV > display size:= 50688*char defined in Core0
extern char *const explode_Split_output__Sobel__3;  // explode_Split_output > Sobel_3 size:= 13376*char defined in Cor
extern uchar *const output_40128__input__0;  // explode_Split_output_output_40128 > Sobel_3_input size:= 13376*uchar de
extern uchar *const output__input_40128__0;  // Sobel_3_output > implode_Merge_input_input_40128 size:= 13376*uchar def
extern char *const explode_Split_output__Sobel__7;  // explode_Split_output > Sobel_7 size:= 13376*char defined in Cor
extern uchar *const output_93632__input__0;  // explode_Split_output_output_93632 > Sobel_7_input size:= 13376*uchar de
extern uchar *const output__input_93632__0;  // Sobel_7_output > implode_Merge_input_input_93632 size:= 13376*uchar def
extern char *const Sobel_0__implode_Merge_input__0;  // Sobel_0 > implode_Merge_input size:= 13376*char defined in Core
extern char *const Sobel_1__implode_Merge_input__0;  // Sobel_1 > implode_Merge_input size:= 13376*char defined in Core
extern char *const Sobel_2__implode_Merge_input__0;  // Sobel_2 > implode_Merge_input size:= 13376*char defined in Core
extern char *const Sobel_4__implode_Merge_input__0;  // Sobel_4 > implode_Merge_input size:= 13376*char defined in Core
extern char *const Sobel_5__implode_Merge_input__0;  // Sobel_5 > implode_Merge_input size:= 13376*char defined in Core
extern char *const Sobel_6__implode_Merge_input__0;                                                              Core
extern uchar *const output__input_0__0;  // Sobel_0                                                              in
extern uchar *const output__input_13376__0;  // Sob                                                             def
extern uchar *const output__input_26752__0;  // Sob                                                             def
extern uchar *const output__input_53504__0;  // Sob                                                             def
extern uchar *const output__input_66880__0;  // Sob                                                             def
extern uchar *const output__input_80256__0;  // Sob                                                             def
extern uchar *const output__input__1;  // implode_M                                                             re0
extern uchar *const output__y__0;  // Merge_output
extern uchar *const u__u__0;  // Read_YUV_u > displ
extern uchar *const v__v__0;  // Read_YUV_v > displ
```

Access to memory

Generated Code - Sobel - Core3

- access to shared memory;
- in charge
- in charge antt chart);

Receive data
Compute sobel*2

```
receiveStart(); // Core0 > Core3: Read_YUV__display__0
receiveEnd(0, 3); // Core0 > Core3: Read_YUV__display__0
receiveStart(); // Core0 > Core3: explode_Split_output__Sobel___3
receiveEnd(0, 3); // Core0 > Core3: explode_Split_output__Sobel___3
sobel(352/*width*/,38/*height*/,output_40128__input__0,output__input_40128__0); // Sobel_3
receiveStart(); // Core0 > Core3: explode_Split_output__Sobel___7
receiveEnd(0, 3); // Core0 > Core3: explode_Split_output__Sobel___7
sobel(352/*width*/,38/*height*/,output_93632__input__0,output__input_93632__0); // Sobel_7
```

Generated Code - Sobel - Core3

```
receiveStart(); // Core0 > Core3: Sobel_0__implode_Merge_input__0
receiveEnd(0, 3); // Core0 > Core3: Sobel_0__implode_Merge_input__0
receiveStart(); // Core1 > Core3: Sobel_1__implode_Merge_input__0
receiveEnd(1, 3); // Core1 > Core3: Sobel_1__implode_Merge_input__0
receiveStart(); // Core2 > Core3: Sobel_2__implode_Merge_input__0
receiveEnd(2, 3); // Core2 > Core3: Sobel_2__implode_Merge_input__0
receiveStart(); // Core0 > Core3: Sobel_4__implode_Merge_input__0
receiveEnd(0, 3); // Core0 > Core3: Sobel_4__implode_Merge_input__0
receiveStart(); // Core1 > Core3: Sobel_5__implode_Merge_input__0
receiveEnd(1, 3); // Core1 > Core3: Sobel_5__implode_Merge_input__0
receiveStart(); // Core2 > Core3: Sobel_6__implode_Merge_input__0
receiveEnd(2, 3); // Core2 > Core3: Sobel_6__implode_Merge_input__0
// Join implode_Merge_input
{
    memcpy(output__input__1+0, output__input_0__0+0, 13376*sizeof(uchar));
    memcpy(output__input__1+13376, output__input_13376__0+0, 13376*sizeof(uch
    memcpy(output__input__1+26752, output__input_26752__0+0, 13376*sizeof(uch
    memcpy(output__input__1+40128, output__input_40128__0+0, 13376*sizeof(uch
    memcpy(output__input__1+53504, output__input_53504__0+0, 13376*sizeof(uch
    memcpy(output__input__1+66880, output__input_66880__0+0, 13376*sizeof(uch
    memcpy(output__input__1+80256, output__input_80256__0+0, 13376*sizeof(uch
    memcpy(output__input__1+93632, output__input_93632__0+0, 13376*sizeof(uch
}
merge(8/*nbSlice*/,352/*width*/,288/*height*/,output__input__1,output__y__0);
```

- Receive all sobel results
  - from other cores only
- Merge
  - Does the memcpy
- then display

Generated Code Sobel - Core1,2

- Access memory
- Receive*2
- Sobel*2
- Send*2

Try out

- Compile & Run Code
  - Open terminal in the Code folder

```
sh CMakeGCC.sh
cd bin/make && make
./Release/sobel
```

Observe performance

Further

- Why poor perf ?
- Instrument the code ?
- Reuse memory ?
- Dynamic scheduling ?
- Coprime integers Fifo input/output?
- Specify an architecture with different cores, having different timings ?

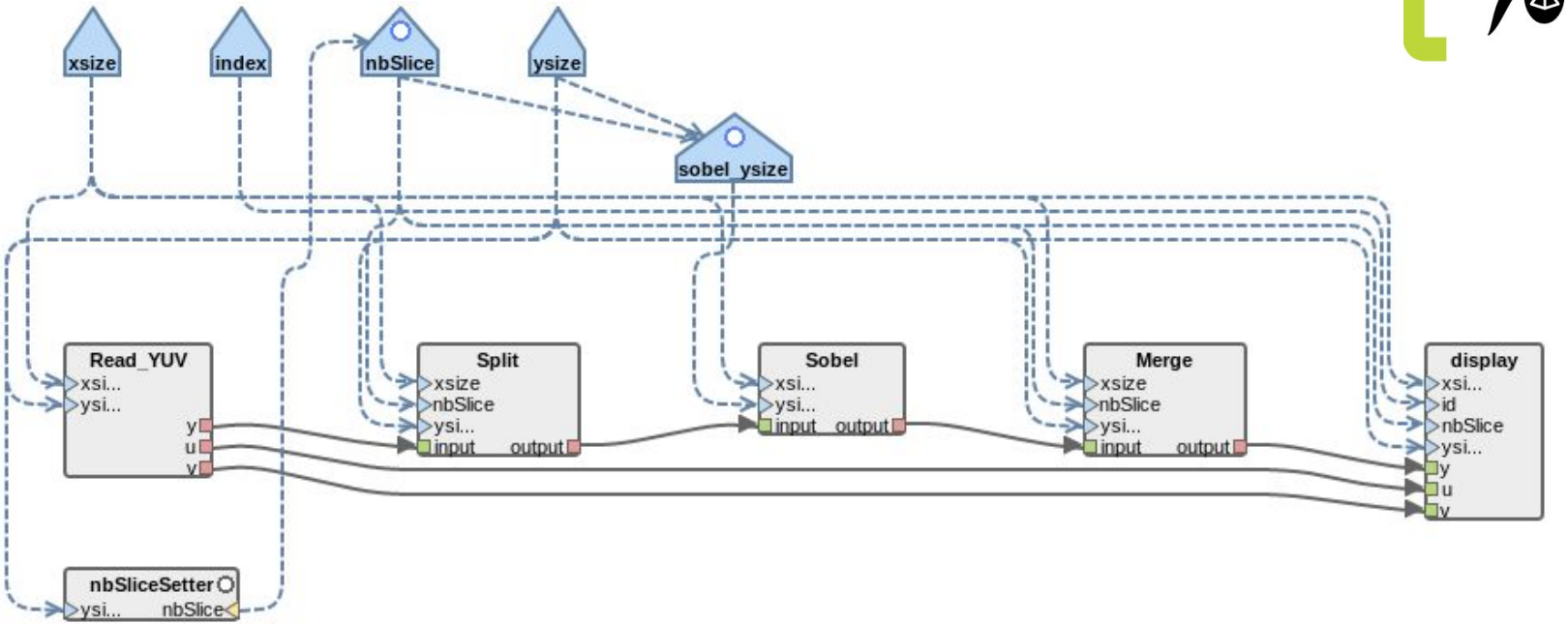
Further

- Why poor perf ? **-> Pipeline (delays), timings**
- Instrument the code ? **-> Accurate schedule input**
- Reuse memory ? **-> memory scripts**
- Dynamic scheduling ? **-> Spider**
- Coprime integers Fifo input/output? **-> EXPLOSION**
- Specify an architecture with different cores, having different timings ? **-> scenario timings**

Further - Dynamic Algos

Dynamic parameters (set by actor)

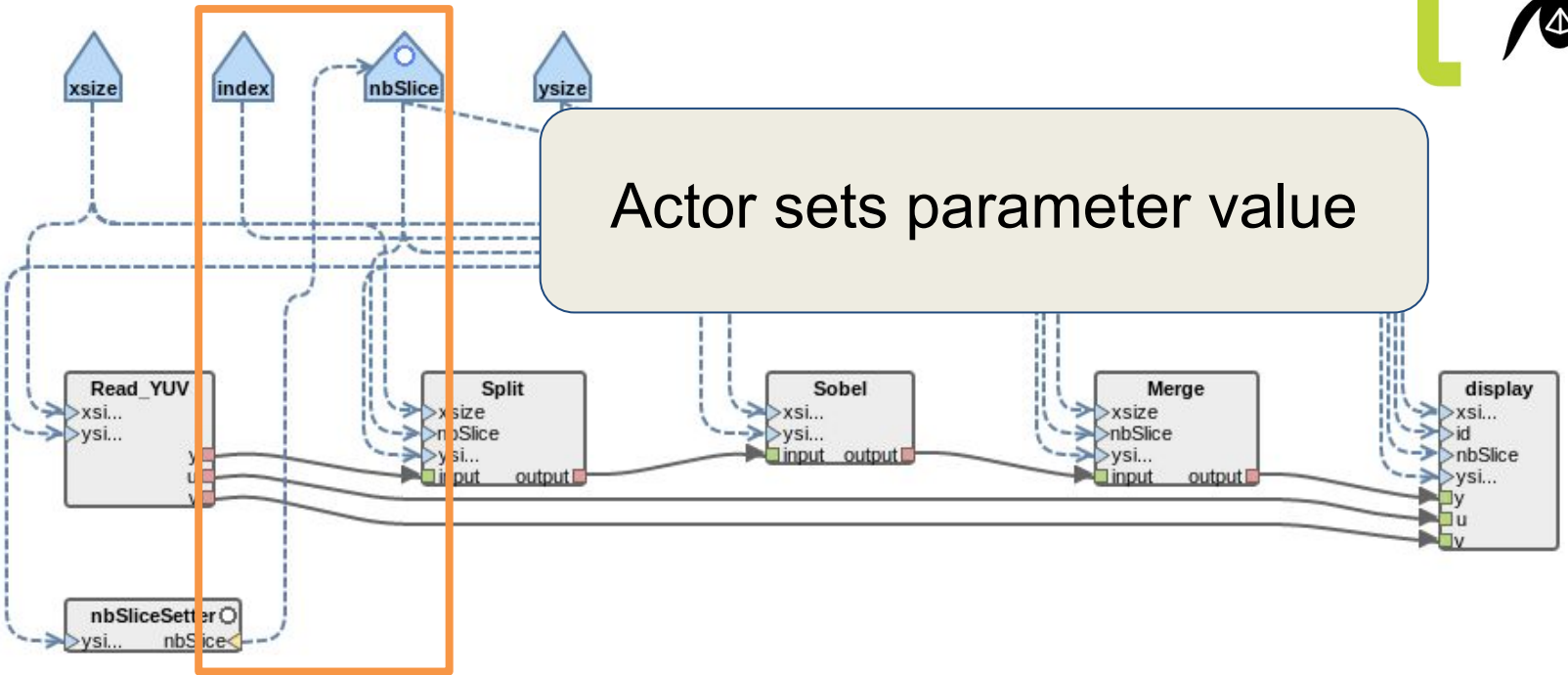
=> Spider



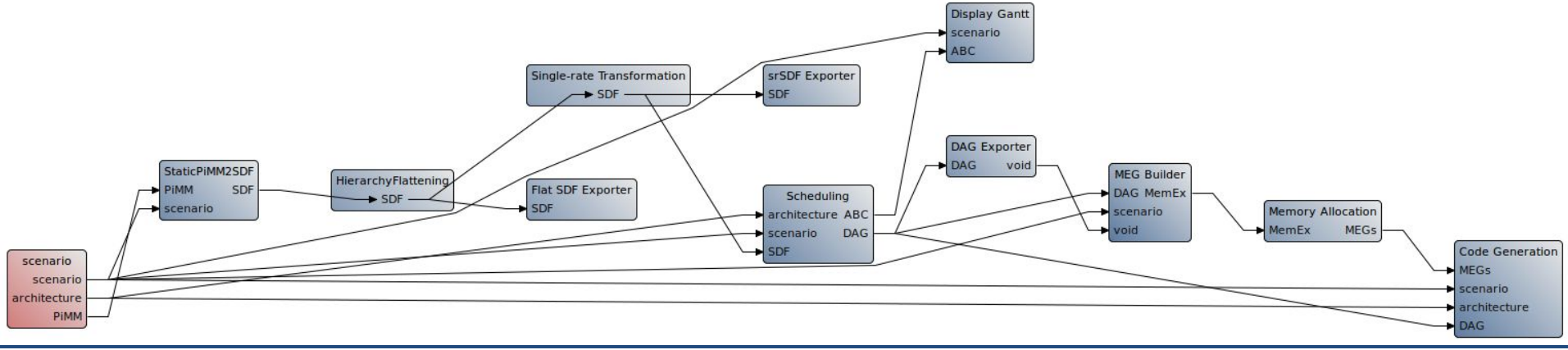
Further - Dynamic Algos

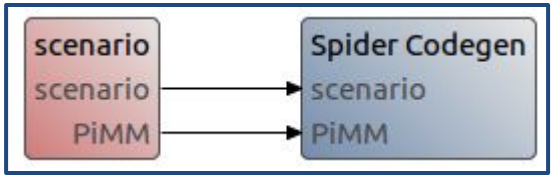
Dynamic parameters (set by actor)

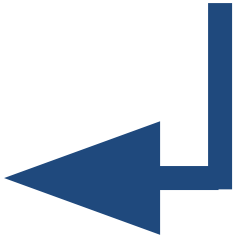
=> Spider



Further - Dynamic Algos







Conclusion

- Rapid prototyping tool
  - SDF MoC + SLAM MoA, Y codesign flow
- Parallelism is not free
  - in term of design (where/when to add split/merge)
  - in term of execution (memcpy everywhere)
- Research Objective (PREESM is a research tool)
  - reduce parallelism design and execution cost
  - tackle memory optimizations further