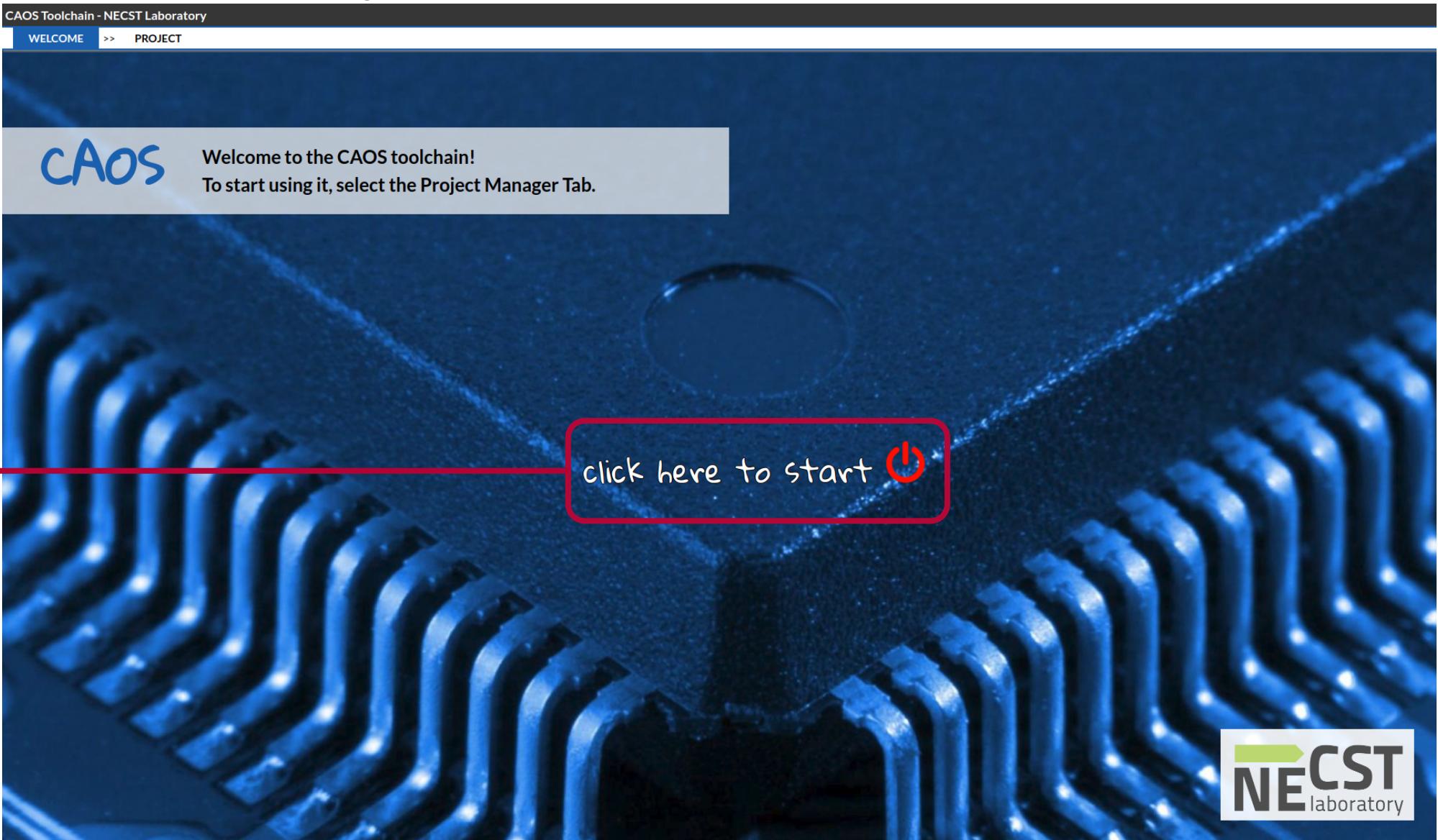


# CAOS tutorial



# Access CAOS at: <http://caos-iccd.necst.it/>



Click on “click here to start” to start the optimization process



# Use CAOS to optimize your application

- Start a new project or load your work right were you left it

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT

Manage your CAOS project ?

New project

Load project

Create a new project

# GUI Overview

**CAOS Toolchain - NECST Laboratory**

WELCOME >> PROJECT >> **IR GENERATION**

**IR Generation**

Phase state: **PROGRESS**

**Input**

Code archive:

Program description:

**CAOS module**

Hostname:  Port:   Status: **ONLINE**

**Output**

Keep track of the **phase** that is being performed using the *navigation window*

# GUI Overview

**CAOS Toolchain - NECST Laboratory**

WELCOME >> PROJECT >> **IR GENERATION**

## IR Generation

Phase state: **PROGRESS**

### Input

Code archive:  

Program description:  

### CAOS module

Hostname:  Port:   Status: **ONLINE**

### Output

Check the **status** of the **phase** on the top-right part of the screen

# GUI Overview

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION

## IR Generation

Phase state: PROGRESS

### Input

Code archive:

Program description:

### CAOS module

Hostname:  Port:   Status: ONLINE

### Output

- One of the main features of CAOS is **Modularity**
  - Every phase is backed up by a CAOS module that can be accessed simply specifying its hostname and port
  - The **CAOS module panel** provides information regarding the status of the specific module

# IR GENERATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION

## IR Generation

Phase state: PROGRESS

### Input

Code archive: [Import...](#)

Program description: [Import...](#)

### CAOS module

Hostname: `m_1.1_from_doxxygen` Port: `5011` [update](#) Status: **ONLINE**

### Output

[Run IR Generation](#)

- **IR Generation** generates the CAOS intermediate representation from the user's source code
- For this phase you need to specify:
  - A zip/tar archive containing the source code of the application to optimize
  - A program description JSON file specifying the source code language and how to compile the application

# Demo applications

The screenshot shows a GitHub repository page for [https://github.com/necst/tutorial\\_iccd17\\_code/](https://github.com/necst/tutorial_iccd17_code/). The page includes a navigation bar with back, forward, and search icons, and a header with a star icon, a fork icon, and other repository details. Below the header are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button, which is highlighted with a red box. The main content area displays a list of commits:

Author	Commit Message	Time Ago
lorenzoditucci	removing unused flags	Latest commit 2b69db6 5 hours ago
SDAccel	removing unused flags	5 hours ago
caos	updated caos vector sum application	a day ago
paper	adding paper for Smith-Waterman acceleration	6 hours ago
README.md	updated README	6 hours ago

- Demo applications are available at: [https://github.com/necst/tutorial\\_iccd17\\_code/](https://github.com/necst/tutorial_iccd17_code/)
- After downloading the repository, you can access the CAOS demo apps under the “caos” folder:
  - Vessel Segmentation
  - Smith-Waterman
- In the following slides we will refer to the *Vessel Segmentation* demo application

# Program Description

```
program-description.json •
```

```
1  {
2      "language": "C++",
3      "supportedCompilers" : [
4          {
5              "compiler": "gcc",
6              "arguments": "-O3"
7          },
8          {
9              "compiler": "llvm",
10             "arguments": "-O3"
11         }
12     ]
13 }
14 }
```

► Provides information regarding the source code:

- Source code language
- Supported compilers
- Arguments needed for compiling

# IR GENERATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION

## IR Generation

Phase state: READY complete

**Input**

Code archive: Import... ✓ Program description: Import... ✓

**CAOS module**

Hostname: m\_1.1\_from\_doxxygen | Port: 5011 | update | Status: ONLINE

**Output**

Run IR Generation

Identified functions

- load\_ppm
- main
- match\_filter
- print\_matrix
- save\_ppm

Program static callgraph

```
graph TD; main --> load_ppm; main --> match_filter; main --> save_ppm; print_matrix
```

- Once the two files have been loaded, by clicking on **Run IR Generation** CAOS:
  - Identifies all the functions within the application
  - Provides a **Program Static Callgraph** highlighting the caller/calle relationships

# IR GENERATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION

## IR Generation

Phase state: **READY** complete

**Input**

Code archive: [Import...](#) ✓ Program description: [Import...](#) ✓

**CAOS module**

Hostname: m\_1.1\_from\_doxxygen | Port: 5011 [update](#) Status: **ONLINE**

**Output**

[Run IR Generation](#)

**Identified functions**

- load\_ppm
- main
- match\_filter
- print\_matrix
- save\_ppm

**Program static callgraph**

```
graph TD; main --> load_ppm; main --> match_filter; main --> save_ppm; print_matrix
```

The callgraph diagram shows the main function at the top, with arrows pointing down to three other functions: load\_ppm, match\_filter, and save\_ppm. The print\_matrix function is shown separately to the right.

Phase state becomes **READY** and we can click on *complete* to move to the next phase

# IR GENERATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> **APPLICABILITY CHECK**

IR Generation

Phase state: COMPLETED edit

**Input**

Code archive: Import... ✓ Program description: Import... ✓

**CAOS module**

Hostname: m\_1.1\_from\_doxxygen Port: 5011 update Status: ONLINE

**Output**

Run IR Generation

Identified functions

- load\_ppm
- main
- match\_filter
- print\_matrix
- save\_ppm

Program static callgraph

```
graph TD; main --> load_ppm; main --> match_filter; main --> save_ppm; print_matrix ---;
```

The callgraph shows the main function at the top, with arrows pointing down to three other functions: load\_ppm, match\_filter, and save\_ppm. The print\_matrix function is shown to the right but has no outgoing arrows, indicating it is a leaf node in the call graph.

Phase state becomes **COMPLETED** and CAOS shows the new phase in the navigation window

# APPLICABILITY CHECK

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK

## Applicability Check

Phase state: PROGRESS

### Input

Architecture description: [Import...](#)

Architectural templates: [Select](#)

### CAOS module

Hostname:  Port:  [update](#) Status: ONLINE

### Output

[Run templates applicability check](#)

- **APPLICABILITY CHECK** verifies which CAOS architectural templates can be used for the given user code and target architecture
- This phase requires:
  - A JSON file providing the target **architecture description**
  - A selection of the **architectural templates** to check

# APPLICABILITY CHECK – Architecture Description

```
architecture-description.json ●  
1 {  
2     "nodeDefinition" : {  
3         "deviceTypes" : { ...  
4             },  
5         "devices" : { ...  
6             },  
7         "connectionTypes" : { ...  
8             },  
9         "connections" : [ ...  
10            ]  
11        },  
12        "system" : { ...  
13        }  
14    }  
15 }
```

- Describes the target architecture on which the application will be executed
- Two levels specification
  - Node description
  - System description
- Let's define the architecture description for a system consisting of a single f1.2xlarge instance...

# APPLICABILITY CHECK – Architecture Description

```
architecture-description.json •  
1 {  
2     "nodeDefinition" : {  
3         "deviceTypes" : {  
4             "f1-fpga" : {  
5                 "type" : "board",  
6                 "vendor" : "Xilinx",  
7                 "partNumber" : "XCVU9P-FLGB2104-2-I"  
8             },  
9             "intel-vcore" : {  
10                 "type" : "cpu",  
11                 "vendor" : "intel",  
12                 "partNumber" : "-"  
13             }  
14         },  
15         "devices" : {  
16             "f1-fpga-instance" : {  
17                 "type" : "f1-fpga"  
18             },  
19             "cpu" : {  
20                 "type" : "intel-vcore",  
21                 "host" : true  
22             }  
23         },  
24         "connectionTypes" : { ...  
25     },  
26         "connections" : [ ...  
27     ]  
28     },  
29     "system" : { ...  
30 }
```

- Defines the type of devices to reference within the node specification
  - Intel CPU
  - Xilinx XCVU9P-FLGB2104-2-I board
- Instantiates the devices and specifies the device acting as “host” for the node

# APPLICABILITY CHECK – Architecture Description

```
architecture-description.json
```

```
1 {  
2     "nodeDefinition" : {  
3         "deviceTypes" : { ...  
4     },  
5         "devices" : { ...  
6     }.  
7         "connectionTypes" : {  
8             "pcie_gen3" : {  
9                 "standard" : "PCIe",  
10                "bandwidth" : "15.75 GB/s",  
11                "duplex" : "full",  
12                "version" : "3.0"  
13            }  
14        },  
15        "connections" : [  
16            {  
17                "source" : "f1-fpga_instance",  
18                "target" : "cpu",  
19                "type" : "pcie_gen3"  
20            }  
21        ]  
22    },  
23    "system" : { ...  
24 }  
25 }  
26  
27 }
```

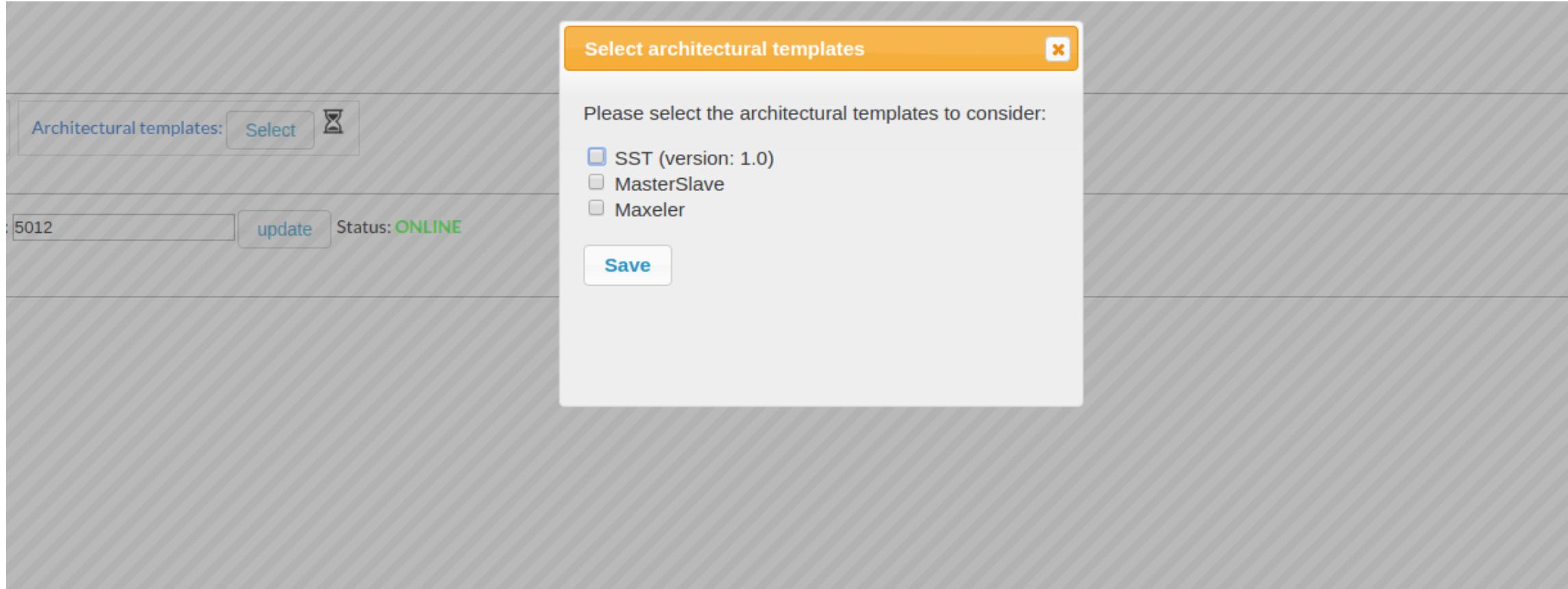
- Specifies the available connections between the devices of the node
- Defines how the devices are interconnected

# APPLICABILITY CHECK – Architecture Description

```
architecture-description.json •  
1 {  
2     "nodeDefinition" : {  
3         "deviceTypes" : { ...  
4     },  
5         "devices" : { ...  
6     },  
7         "connectionTypes" : { ...  
8     },  
9         "connections" : [ ...  
10    ]  
11    },  
12    "system" : {  
13        "nodes" : ["f1-fpga_node"],  
14        "connectionTypes" : {},  
15        "connections" : []  
16    }  
17}  
18}  
19}
```

- Defines the number of nodes available within the system and how they are interconnected
- For the f1.2xlarge instance we simply define a single-node system

# APPLICABILITY CHECK – Architectural Template



- Select which architectural template(s) you want CAOS to consider for your application
  - Click Save
  - Click on *Run templates applicability check*

# APPLICABILITY CHECK – Result

## Architectural Templates Validation:

### MasterSlave

Status: **Template is supported!**

#### Architecture report:

Supported devices:

- f1-fpga\_instance

#### Functions report:

##### load\_ppm

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'load\_ppm' call: [fopen]

##### main

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'main' call: [printf]

##### match\_filter

- Hardware acceleration: **yes**

##### print\_matrix

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'print\_matrix' call: [printf]

##### save\_ppm

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'save\_ppm' call: [fopen]

### Maxeler

Status: **Template not supported.**

Reason: the system does not provide a supported device, or matches the specified connection rules

Additional info: supported devices: board\_Vectis

- The *Architectural Template Validation* shows which architectural templates are being supported, and on which devices

# APPLICABILITY CHECK – Result

Architectural Templates Validation:

## MasterSlave

Status: **Template is supported!**

### Architecture report:

Supported devices:

- f1-fpga\_instance

### Functions report:

#### load\_ppm

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'load\_ppm' call: [fopen]

#### main

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'main' call: [printf]

#### match\_filter

- Hardware acceleration: **yes**

#### print\_matrix

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'print\_matrix' call: [printf]

#### save\_ppm

- Hardware acceleration: **no**
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'save\_ppm' call: [fopen]

## Maxeler

Status: **Template not supported.**

Reason: the system does not provide a supported device, or matches the specified connection rules

Additional info: supported devices: board\_Vectis

- Provides information on which functions are **suitable** for a **hardware acceleration**

# PROFILING

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING

## Profiling

### Input

Profiling dataset: [Add...](#) 

### CAOS module

Hostname: m\_1.3\_cpp\_perf\_prof Port: 5013 [update](#) Status: **ONLINE**

### Output

[Profile datasets](#)

### Import Dataset



Dataset name:

Please select a dataset archive:

[Scegli file](#) Nessun file selezionato

Command line arguments:

NOTE: use %%DATASET\_DIR%% to refer to the directory where the archive will be extracted

[SUBMIT](#)

- Click “Add...” on *Profiling Dataset* to provide a dataset for the profiling phase:
  - Choose a *Name*
  - Upload an Archive containing the dataset (input files for the application)
  - Command line arguments necessary for the execution
- **N.B.** If the source code auto-generates the dataset, the archive and arguments might not be needed

# PROFILING

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING

## Profiling

### Input

Profiling dataset: [Add...](#)



### CAOS module

Hostname: m\_1.3\_cpp\_perf\_prof Port: 5013 [update](#) Status: **ONLINE**

### Output

[Profile datasets](#)

**Import Dataset** X

Dataset name:

Please select a dataset archive: [Scegli file](#) Nessun file selezionato

Command line arguments:

NOTE: use %%DATASET\_DIR%% to refer to the directory where the archive will be extracted

**SUBMIT**

Click **SUBMIT** and **Profile datasets** to start code profiling

# PROFILING

## Output

Profile datasets

Dataset: NO\_DATASET

Function	Self time %	Total time %
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])	98.96%	98.96%
main(int, char *[])	0.21%	99.25%
Overall external calls	0.84%	n.a.

- Profiling result
  - Identifies the most **computationally intensive** functions of the application
  - Provides percentages regarding **self** and **total execution time**

# PARTITIONING

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING >> PARTITIONING

## HW / SW partitioning

### CAOS module

Hostname: m\_1.4\_default Port: 5014  Status: ONLINE

### Output

Function	Profiling Data						HW / SW partitioning HW acceleration per architectural template	
	Self time %			Total time %				
	min	avg	max	min	avg	max		
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])	98.96%	98.96%	98.96%	98.96%	98.96%	98.96%	- let partitioner decide - ▾	
main(int, char *[])	0.21%	0.21%	0.21%	99.25%	99.25%	99.25%	Software	
Overall external calls	0.84%	0.84%	0.84%	n.a.	n.a.	n.a.		

- For each function that is candidate for **Hardware Acceleration** it is possible to decide to force a hardware or software implementation, or to let CAOS decide

# PARTITIONING

## HW / SW partitioning

### CAOS module

Hostname: m\_1.4\_default Port: 5014  Status: ONLINE

### Output

Function	Profiling Data						HW / SW partitioning HW acceleration per architectural template	
	Self time %			Total time %				
	min	avg	max	min	avg	max		
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])	98.96%	98.96%	98.96%	98.96%	98.96%	98.96%	Hardware	
main(int, char *[])	0.21%	0.21%	0.21%	99.25%	99.25%	99.25%	Software	
Overall external calls	0.84%	0.84%	0.84%	n.a.	n.a.	n.a.		

- After clicking *Run partitioning* CAOS automatically selects which functions to accelerate on hardware

# FUNCTIONS OPTIMIZATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING >> PARTITIONING >> FUNCTIONS OPTIMIZATION

Functions optimization

Code archives

Architectural template: MasterSalve

 Initial Code 

[Start optimization](#)

Phase state: **READY** complete

- Within the **Functions Optimization** phase, we can specify the version of the code we want to work on by clicking on the corresponding folder
- Initially, only one version of the code is available
- After selecting the code version, we can either decide to:
  - a) Click *complete* and move to the next phase
  - b) Click on *Start Optimization* to start optimizing the code

**Functions optimization**

Code archives

Architectural template: MasterSalve

 Initial Code 

[Start optimization](#)

# FUNCTIONS OPTIMIZATION – Pre-Opt IR Generation

**Functions optimization**

**CANCEL** **PRE-OPT IR GENERATION**

**CAOS module**

Hostname: m\_1.1\_from\_dxygen Port: 5011 **update** Status: **ONLINE**

**Output**

**Generate IR**

**Identified functions**

```
load_ppm(char *, unsigned char[1080][1440], int)
main(int, char **)
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])
print_matrix(short[1080][1440])
save_ppm(char *, unsigned char[1080][1440])
```

- During **Function Optimization** there are multiple sub-phases
- This first one, generates an *intermediate representation* of the code before optimizing it

# FUNCTIONS OPTIMIZATION – Static Code Analysis

## Functions optimization

CANCEL    PRE-OPT IR GENERATION    >>    STATIC CODE ANALYSIS

**CAOS module**

Hostname: m\_2.1\_default    Port: 5021    **update**    Status: **ONLINE**

**Run static code analysis**

**Functions metrics**

Function	Property	Value
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])	averageLatency	7002323881
	bestLatency	7002323881
	worstLatency	7002323881

- The **Static Code Analysis** phase provides a set of properties (such as expected latency) for all the functions that are candidates for hardware acceleration

# FUNCTIONS OPTIMIZATION – Hardware Estimation

## Functions optimization

Phase state: PROGRESS

CANCEL

PRE-OPT IR GENERATION

>>

STATIC CODE ANALYSIS

>>

HARDWARE ESTIMATION

CAOS module

Phase state: READY complete

Hostname: m\_2.2\_default

Port: 5022

update

Status: ONLINE

Run hardware estimation

Hardware resources estimation

Function	Device	Resource Type	Quantity
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])	f1-fpga	BRAM_18K	3 (0.07%)
		DSP48E	7 (0.10%)
		FF	1024 (0.04%)
		LUT	1597 (0.14%)
		URAM	0 (0.00%)

- By clicking on *Run hardware estimation* CAOS estimates the amount of **resources** needed for implementing the functions on the FPGA

# FUNCTIONS OPTIMIZATION – Performance Estimation

## Functions optimization

Phase state: PROGRESS

CANCEL   PRE-OPT IR GENERATION   >>   STATIC CODE ANALYSIS   >>   HARDWARE ESTIMATION   >>   **PERFORMANCE ESTIMATION**

CAOS module   Phase state: READY complete

Hostname: m\_2.3\_default   Port: 5023   update   Status: ONLINE

[Run performance estimation](#)

**Functions performance estimation and suggested optimization**

**Top function:** match\_filter(unsigned char[1080][1440], unsigned char[1080][1440])

**Current function implementation**

**Performance estimation**  
Execution time: 35.012 s  
Clock frequency: 200 MHz

**Hardware estimation**

f1-fpga	BRAM_18K	3 (0.07%)
	DSP48E	7 (0.10%)
	FF	1024 (0.04%)
	LUT	1597 (0.14%)
	URAM	0 (0.00%)

**Optimization 1)**

**Optimization type:** on\_chip\_full\_caching

**Performance estimation**  
Execution time: 29.056 s  
Clock frequency: 200 MHz

**Hardware estimation**

f1-fpga	BRAM_18K	1523 (35.25%)
	DSP48E	7 (0.10%)
	FF	1211 (0.05%)
	LUT	2211 (0.19%)
	URAM	0 (0.00%)

**Parameters:**

fpga_to_host_copy:	[0,1]
host_to_fpga_copy:	[0,1]

**Optimization 2)**

**Optimization type:** pipelining

**Performance estimation**  
Execution time: 13.750 s  
Clock frequency: 200 MHz

**Hardware estimation**

f1-fpga	BRAM_18K	3 (0.07%)
	DSP48E	7 (0.10%)
	FF	1274 (0.05%)
	LUT	1937 (0.16%)
	URAM	0 (0.00%)

**Parameters:**

ll:	1
relativeForLine:	15

➤ After hardware estimation, CAOS estimates **performance** suggesting potential optimizations

# FUNCTIONS OPTIMIZATION – Code Optimization

## Functions optimization

Phase state: PROGRESS

CANCEL

PRE-OPT IR GENERATION

>>

STATIC CODE ANALYSIS

>>

HARDWARE ESTIMATION

>>

PERFORMANCE ESTIMATION

>>

CODE OPTIMIZATION

CAOS module

Phase state: READY complete

Hostname: m\_2.4\_default

Port: 5024

update

Status: ONLINE

### Input

#### Function

match\_filter(unsigned char[1080][1440], unsigned char[1080][1440])

#### Optimization

Optimization 1 (on\_chip\_full\_caching)

### Output

Apply code optimizations

### Optimization Report

function	optimization applied
load_ppm(char *, unsigned char[1080][1440], int)	NO
main(int, char *[])	NO
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])	YES
print_matrix(short[1080][1440])	NO
save_ppm(char *, unsigned char[1080][1440])	NO

- Within **Code Optimization** sub-phase, it is possible to select among the suggested optimizations
- By clicking *Apply code optimizations* the code will be modified accordingly

# FUNCTIONS OPTIMIZATION – Post-Opt IR Generation

CANCEL    PRE-OPT IR GENERATION    >>    STATIC CODE ANALYSIS    >>

**POST-OPT IR GENERATION**

**CAOS module**

Hostname: m\_1.1\_from\_dxygen    Port: 5011    **update**    Status: **ONLINE**

**Output**

**Regenerate IR**

**Identified functions**

```
load_ppm(char *, unsigned char[1080][1440], int)
main(int, char [])
match_filter(unsigned char[1080][1440], unsigned char[1080][1440])
print_matrix(short[1080][1440])
save_ppm(char *, unsigned char[1080][1440])
```

- After having applied one of the suggested code optimizations the IR might have changed
- The Post-Opt IR Generation sub-phase allows to regenerate the IR.

# FUNCTIONS OPTIMIZATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING >> PARTITIONING >> FUNCTIONS OPTIMIZATION

## Functions optimization

Phase state: **READY** complete

### Code archives

#### Architectural template: MasterSalve



[Start optimization](#)

- The optimization process is complete, we can either:
  - try to apply more optimizations by selecting the new code version and clicking on *Start Optimization*
  - select the desired code version and move to the implementation phase

# IMPLEMENTATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING >> PARTITIONING >> FUNCTIONS OPTIMIZATION >> IMPLEMENTATION

## Implementation

Phase state: **READY** complete

### CAOS module

Hostname:  Port:   Status: **ONLINE**

### Output



Implementation archive

- During the **Implementation** phase CAOS will produce the host, kernel file and Makefile for SDAccel
- Once the phase is completed, click on the Implementation Archive to download the code
- After running the implementation, the output archive is also accessible on the server at:

***~/Documents/CAOS\_outputs***

# Emulate / Build the final application

## ➤ Upload the CAOS archive to an AWS instance

```
- scp -i <pem file> <CAOS-ARCHIVE>.zip centos@<public_dns_entry>:~/
```

## ➤ SSH to the AWS Instance and load the SDAccel settings

```
- ssh -i <pem file> centos@<public_dns_entry>
```

```
- cd $AWS_FPGA_REPO_DIR
```

```
- source sdaccel_setup.sh
```

```
- source /opt/Xilinx/SDx/2017.1.op/settings64.sh
```

## ➤ Unzip the CAOS archive

```
- cd ~/
```

```
- unzip <CAOS-ARCHIVE>.zip
```

```
- cd output
```

## ➤ Use the Makefile to run HW / SW emulation or build the application

```
- make emulation TARGET=sw_emu
```

```
- make emulation TARGET=hw_emu
```

```
- make build TARGET=hw
```

# Save the CAOS project

The screenshot shows the CAOS Toolchain interface. At the top, a navigation bar lists: WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING >> PARTITIONING >> FUNCTIONS OPTIMIZATION >> IMPLEMENTATION. Below this, a sub-menu titled "Manage your CAOS project" is displayed, featuring a red-bordered button labeled "Save current project". A second button below it is labeled "Close current project".

- CAOS flow is over, remember to save your project for future use, and then close it.

