

CAOS: CAD as an Adaptable OpenPlatform Service

31/08/2018 - FPL2018, Dublin IE

Marco Rabozzi <marco.rabozzi@polimi.it>

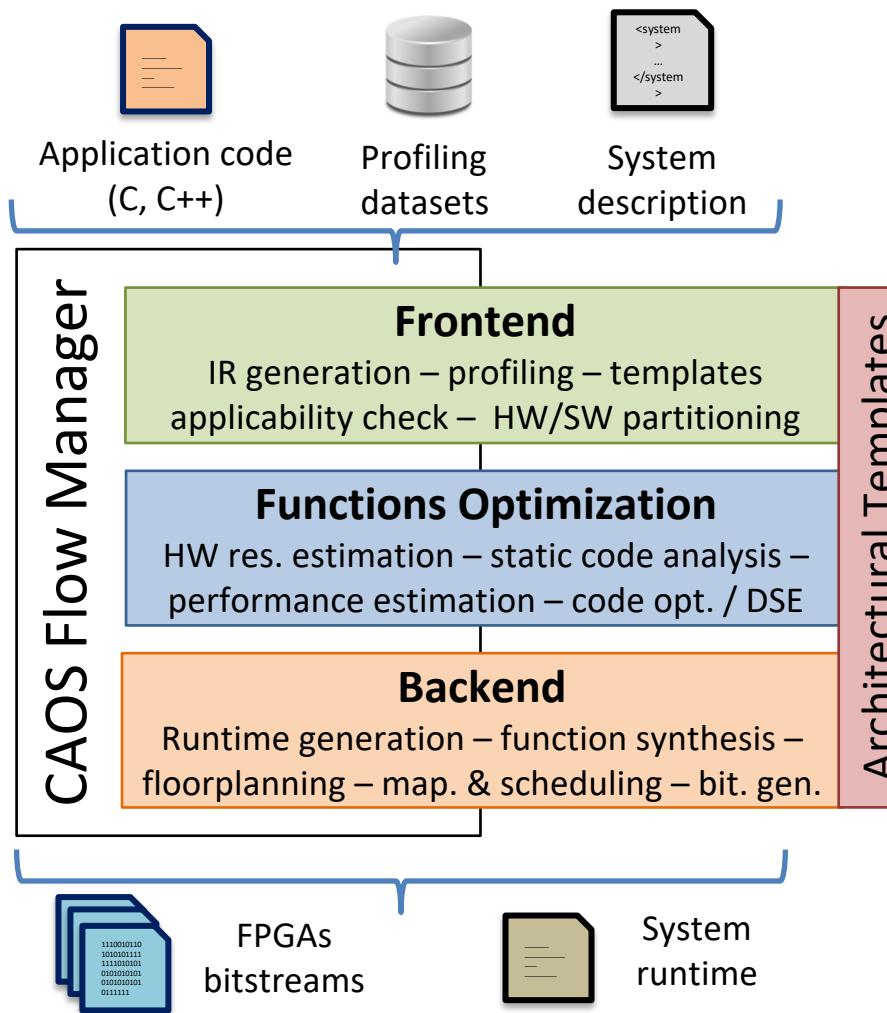
Lorenzo Di Tucci <lorenzo.ditucci@polimi.it>

Marco D. Santambrogio <marco.santambrogio@polimi.it>



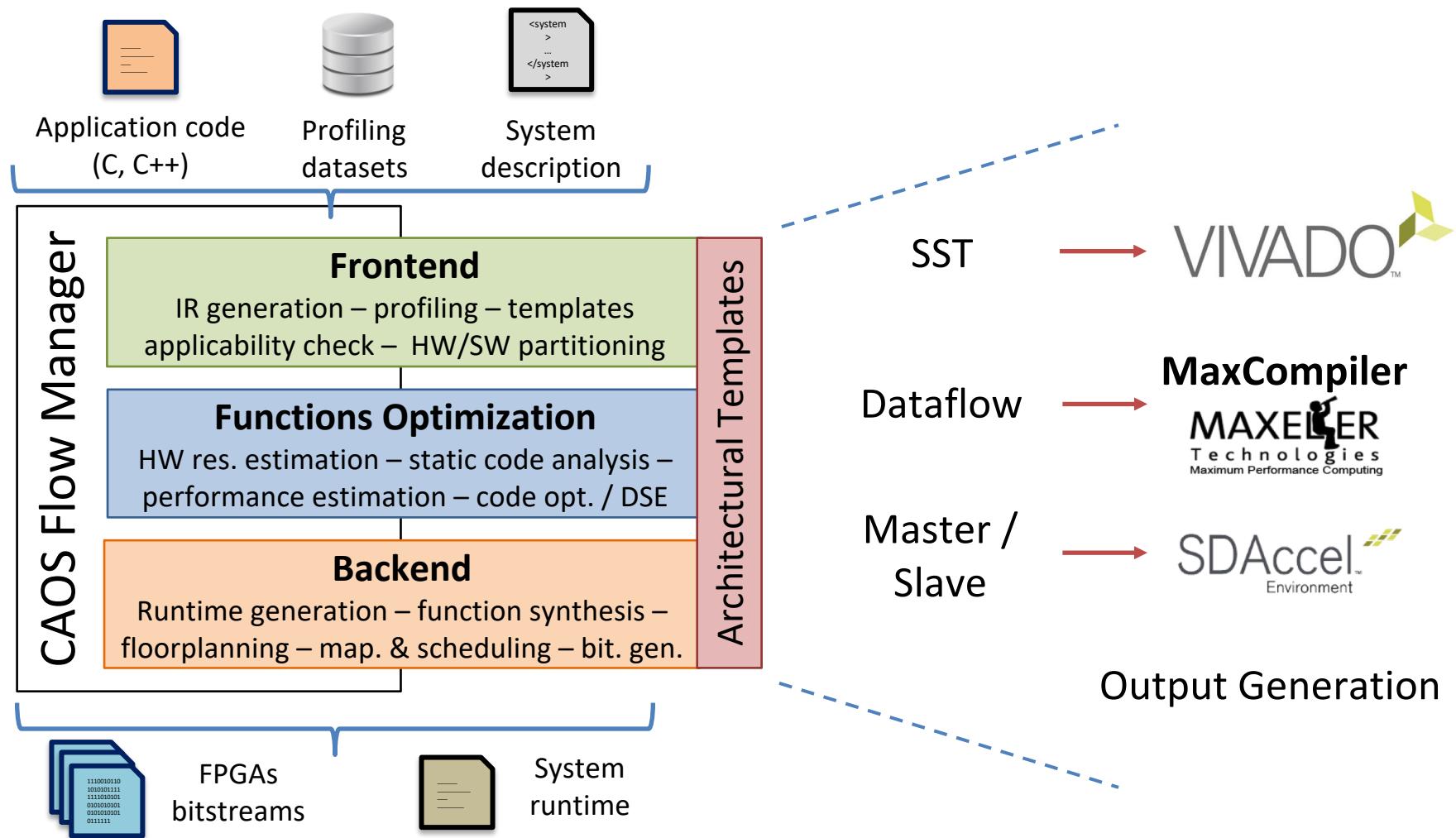


- guide the application designer in the implementation of efficient hardware-software solutions for high performance FPGA-based systems
- promote open research by allowing external researchers to easily integrate and benchmark their algorithms

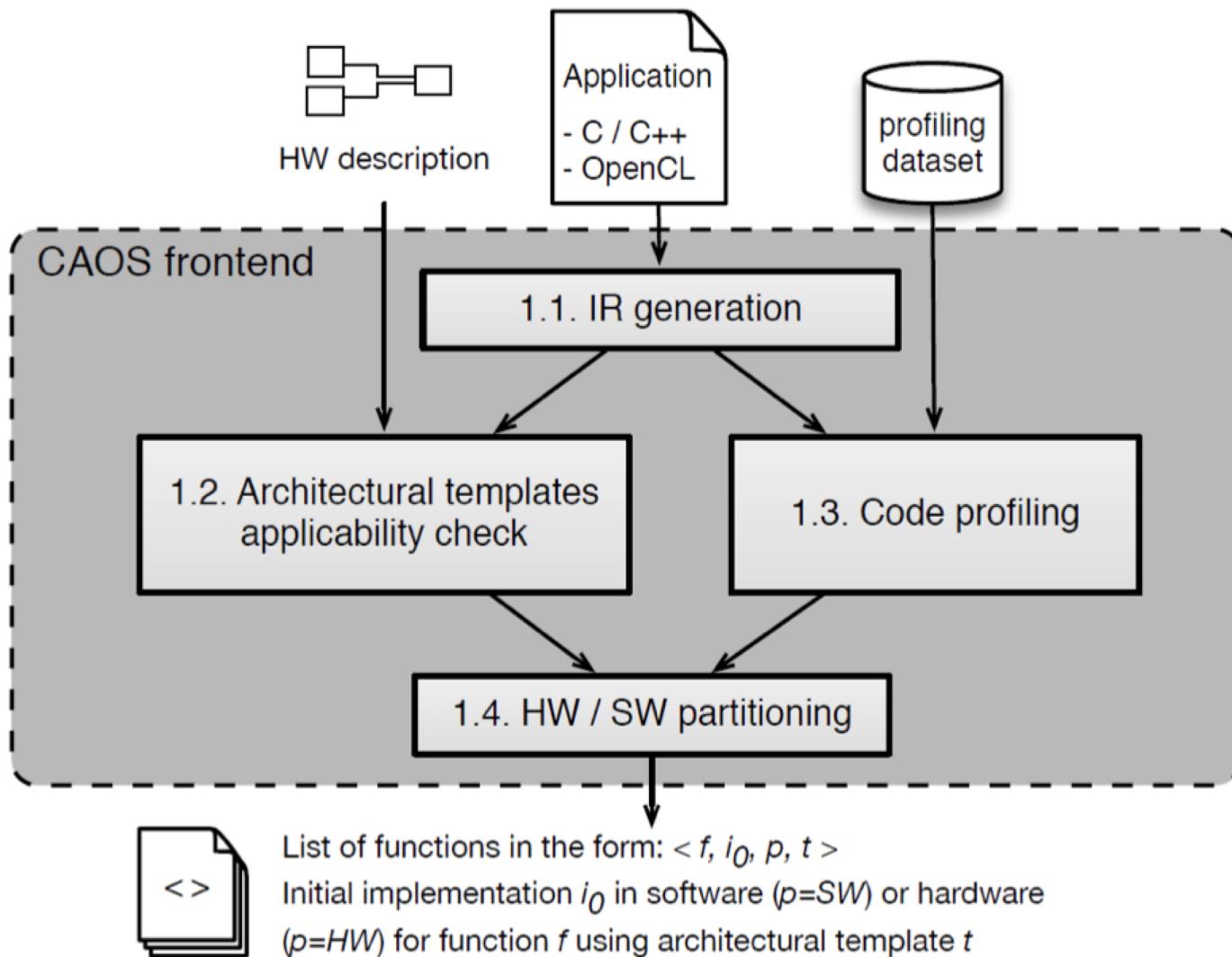


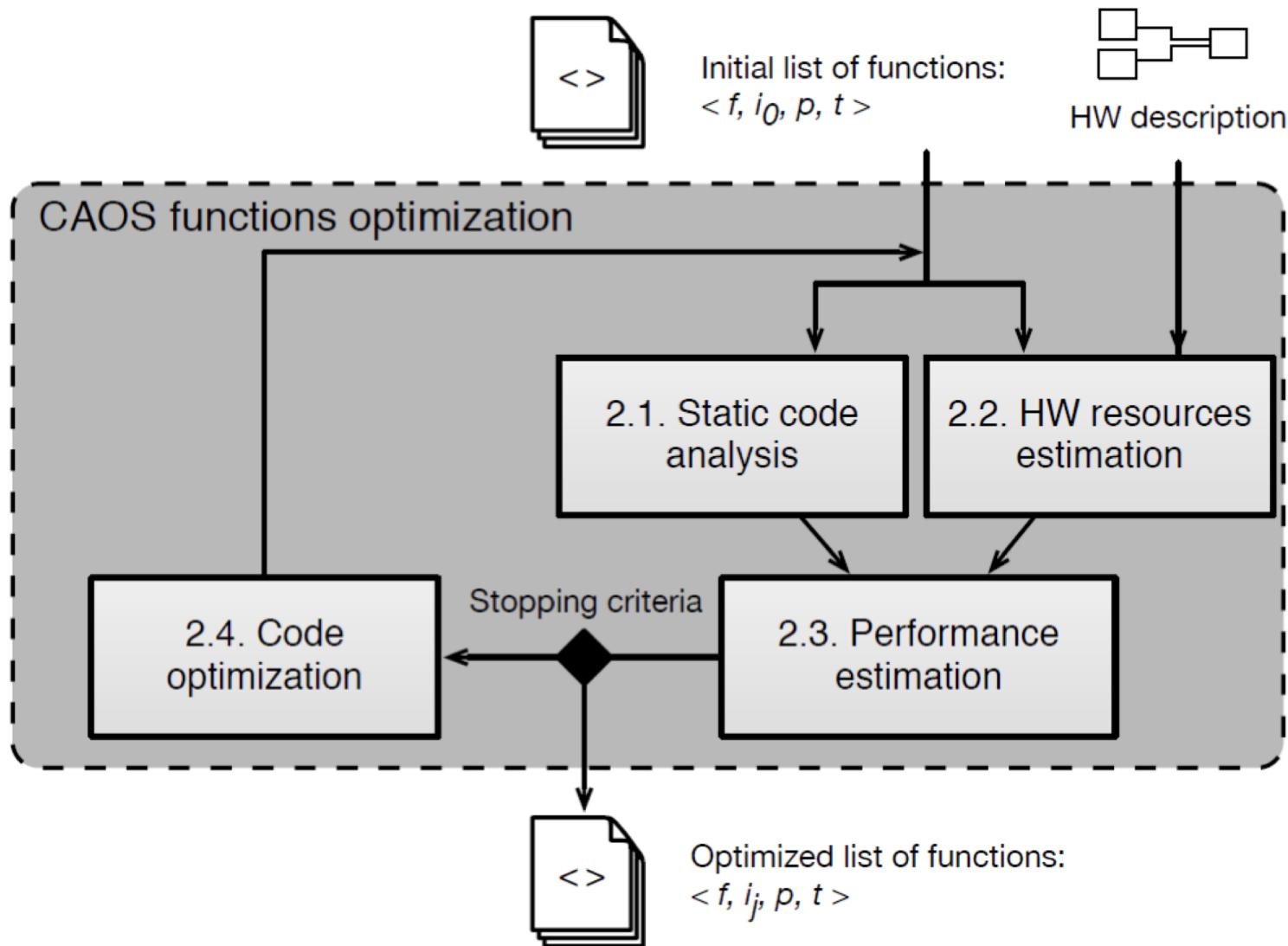


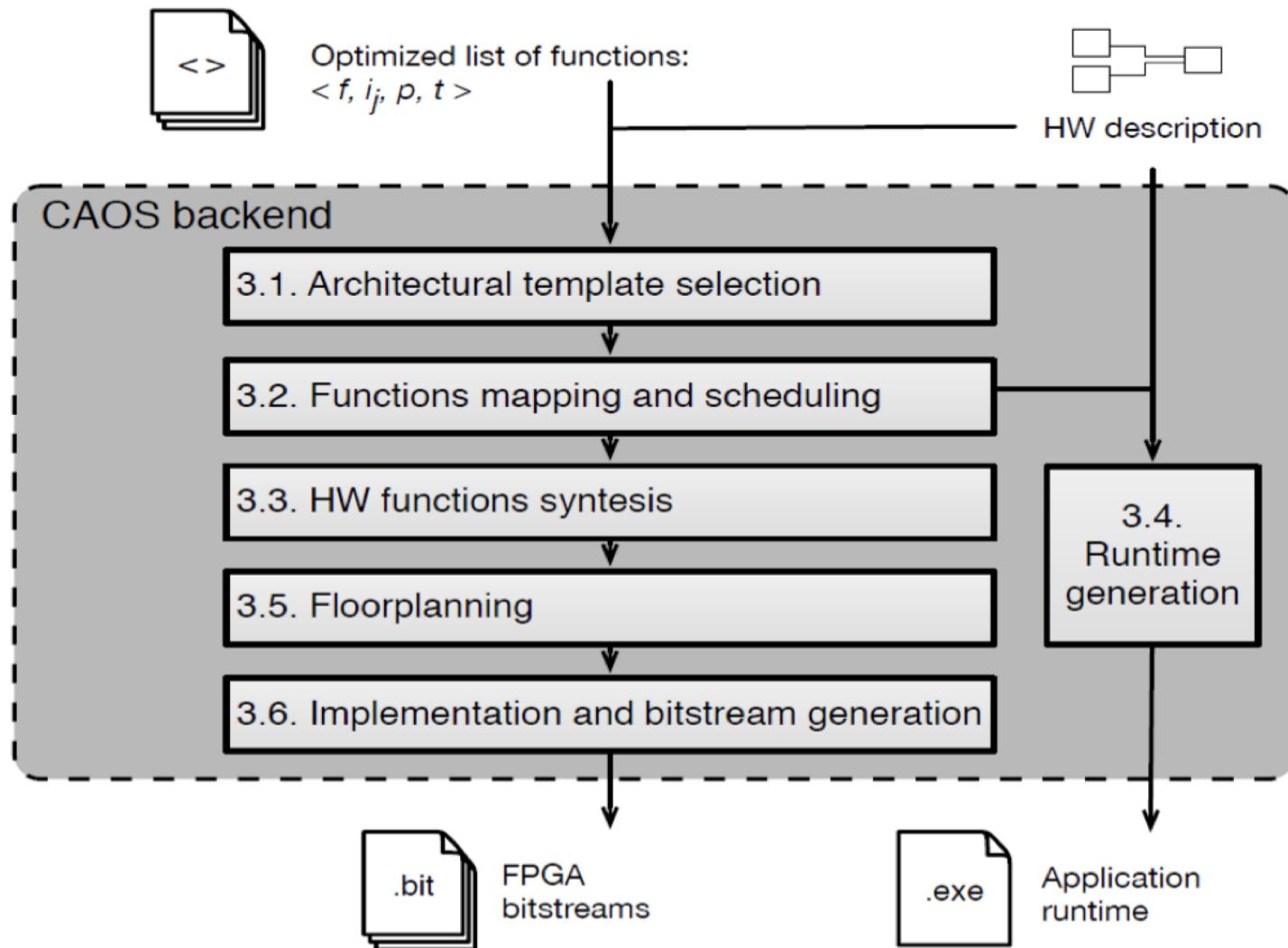
The proposed CAOS framework

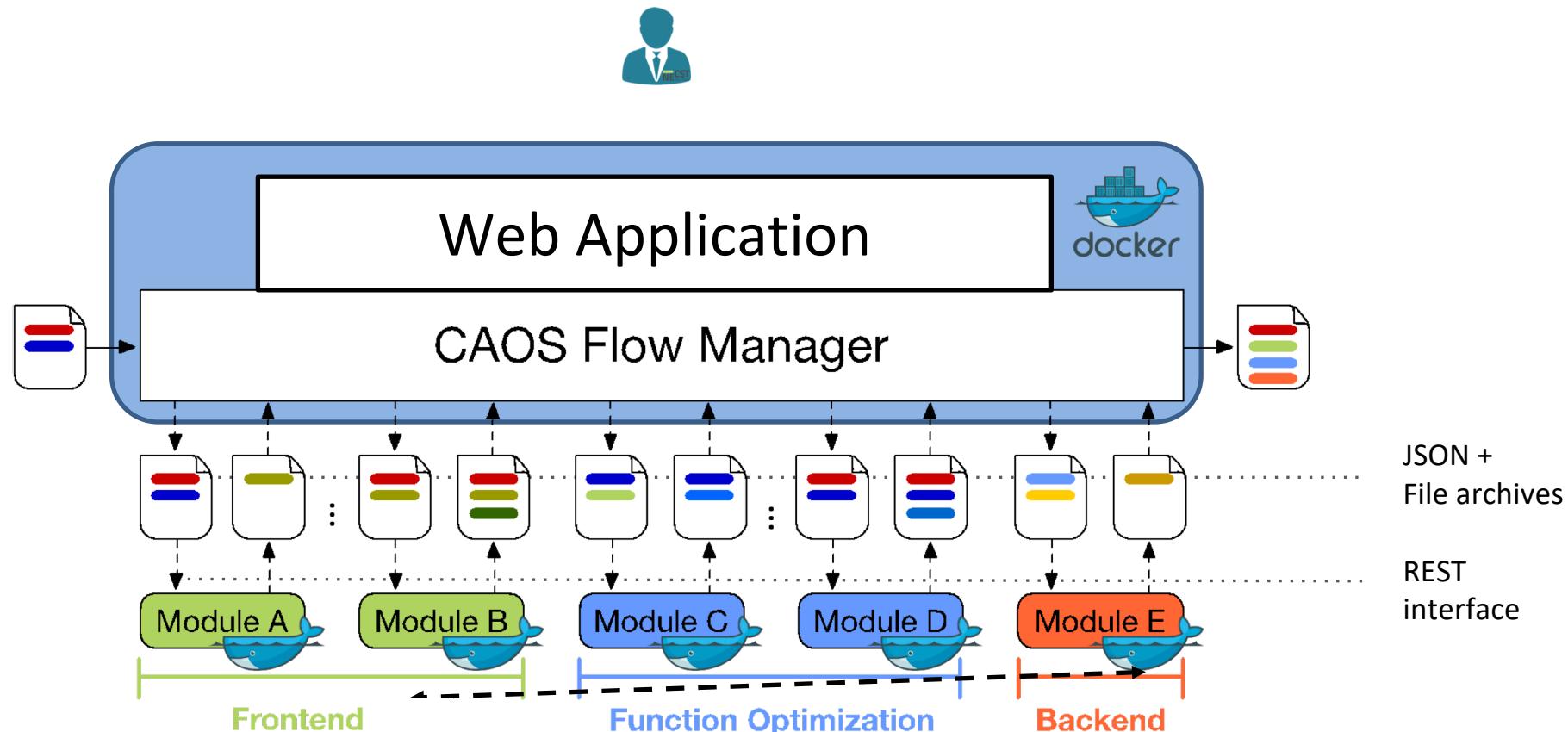


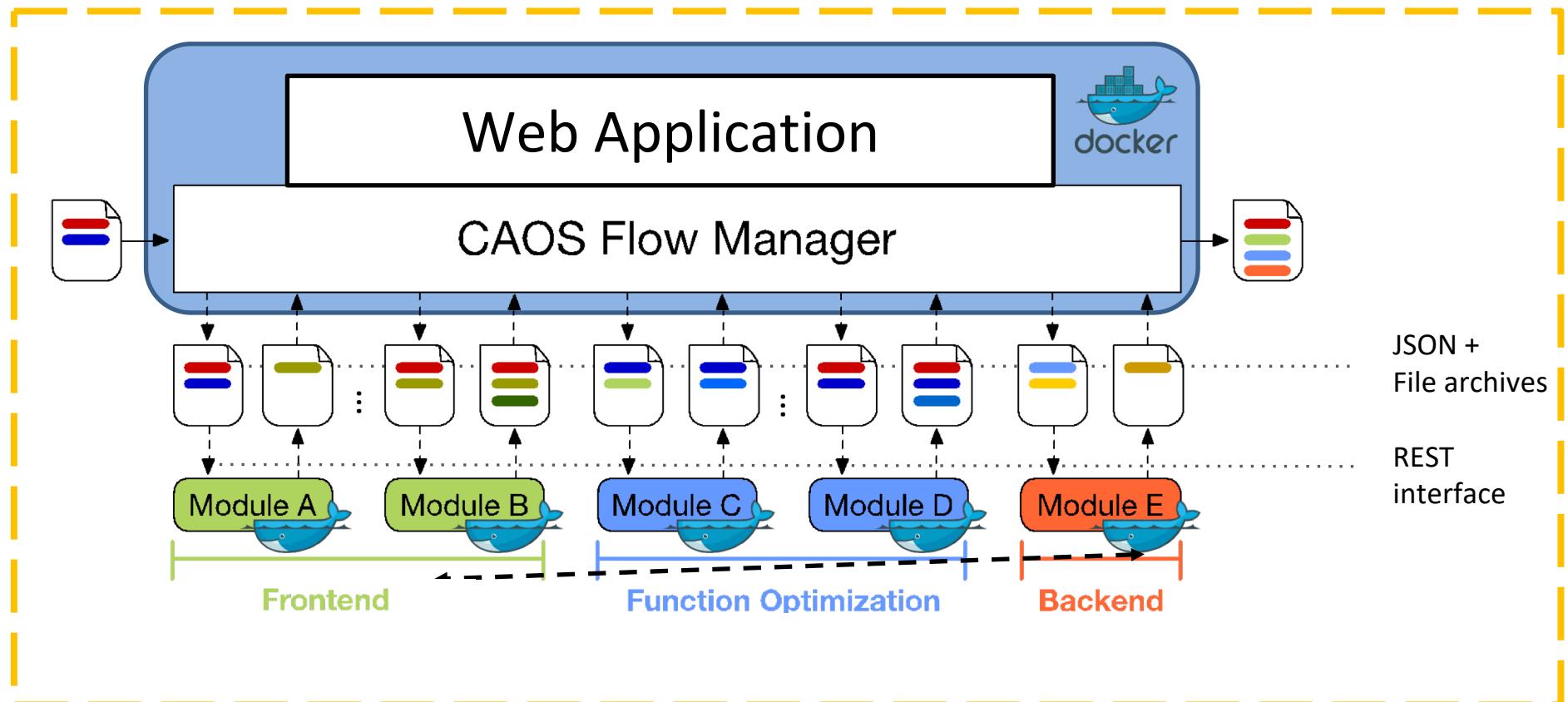
- **Narrow the Design Space Exploration (DSE) for the acceleration**
 - Well defined set of potential optimizations
 - Constraints the classes of supported algorithms
- **Enable more accurate estimations**
 - Hardware resource requirements
 - Operational intensity estimation











- **SST (Single Streaming Timestep)**
 - Scientific applications with a regular execution flow in which
 - a N-dimensional data structure is updated over time
 - Example: Heat flow simulation, Gauss-Seidel linear systems solver
- **Dataflow**
 - Applications with a static compute graph where the same computation is repeated for multiple input items
 - Example: Asian Option Pricing, Image processing filters
- **Master / Slave**
 - More general applications with a regular execution flow, ideal for working set that can be effectively tiled
 - Example: N-Body Physics Simulation, Retinal Vessel Segmentation

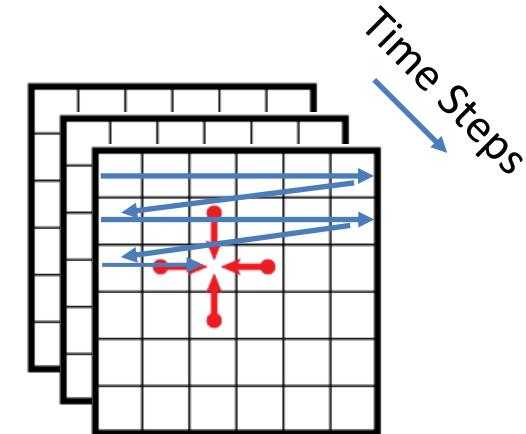


- **SST (Single Streaming Timestep)**
 - Scientific applications with a regular execution flow in which
 - a N-dimensional data structure is updated over time
 - Example: Heat flow simulation, Gauss-Seidel linear systems solver
- **Dataflow**
 - Applications with a static compute graph where the same computation is repeated for multiple input items
 - Example: Asian Option Pricing, Image processing filters
- **Master / Slave**
 - More general applications with a regular execution flow, ideal for working set that can be effectively tiled
 - Example: N-Body Physics Simulation, Retinal Vessel Segmentation

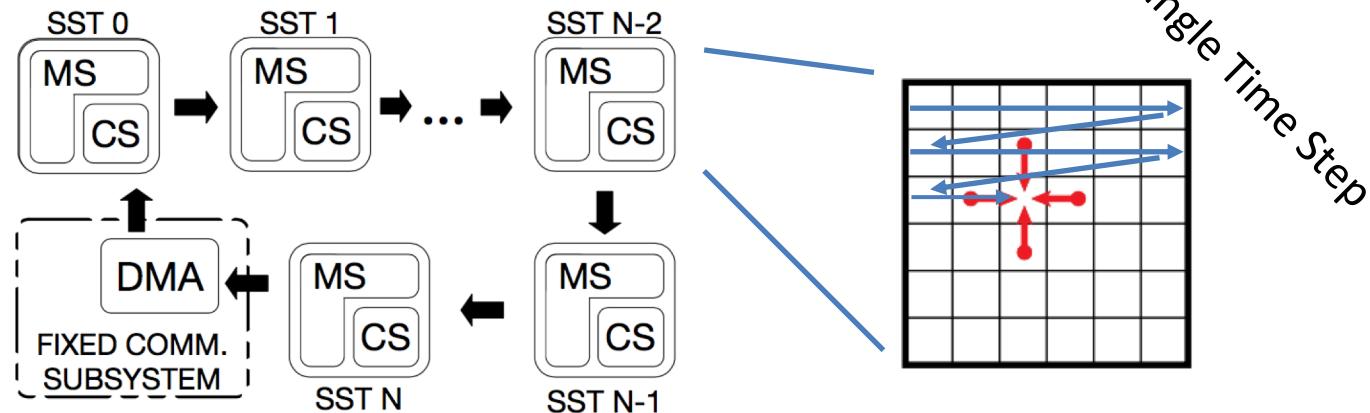
- Iterative Stencil Loop (ISL) Algorithm

Algorithm 1 Generic ISL Algorithm

```
for  $t \leq TimeSteps$  do
    for all points  $p$  in matrix  $M$  do
         $p \leftarrow stencil(p)$ 
```

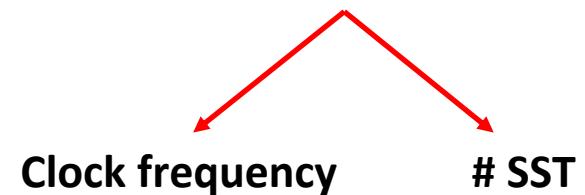


- FPGA-based Stencil Time-Step (SST) Architecture [1]



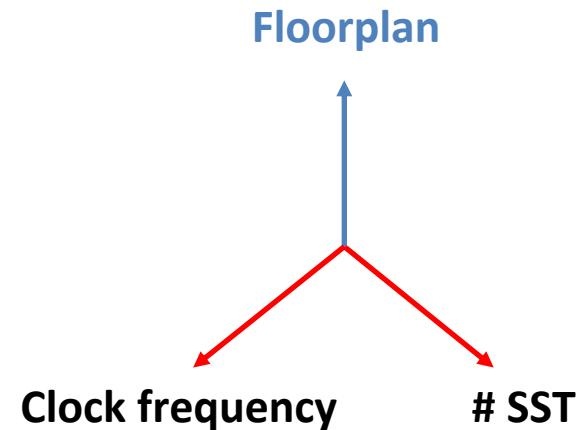
[1] Natale, G., Stramondo, G., Bressana, P., Cattaneo, R., Sciuto, D., & Santambrogio, M. D.. A polyhedral model-based framework for dataflow implementation on FPGA devices of Iterative Stencil Loops. In Computer-Aided Design (ICCAD), 2016 International Conference on (pp. 1-8). IEEE.

- Problem:
 - Identify an optimal implementation of an SST-based design on a target FPGA



- Problem:

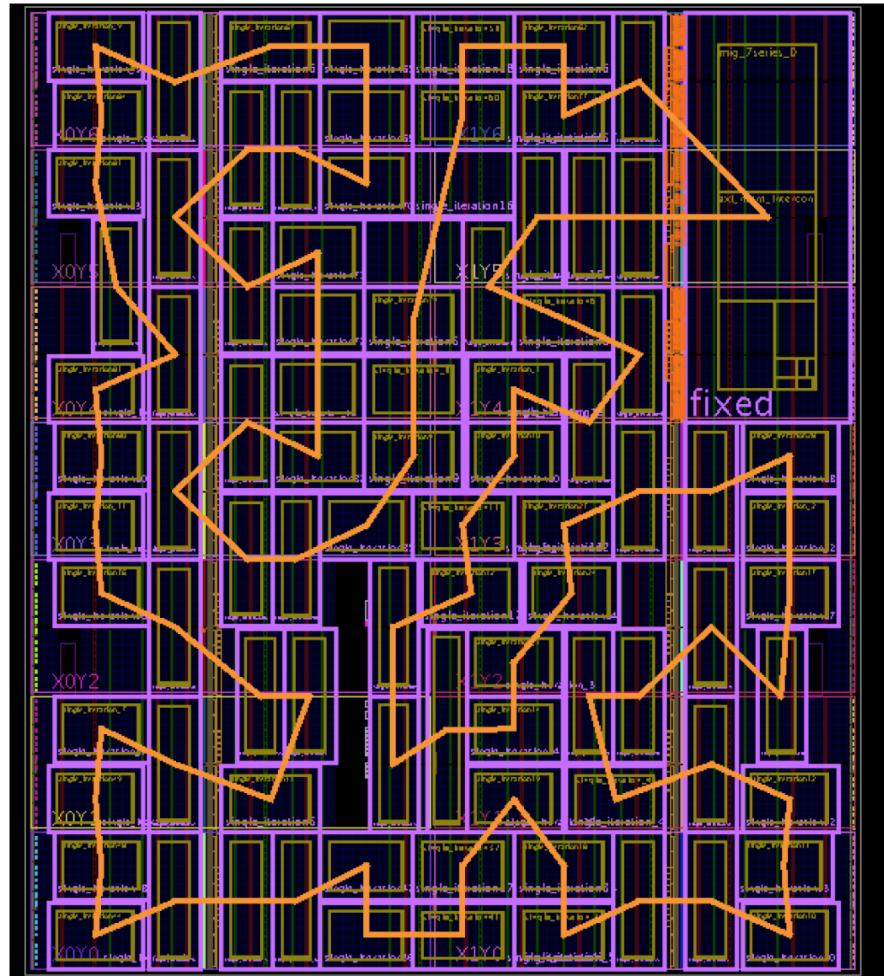
- Identify an optimal implementation of an SST-based design on a target FPGA



- Proposed Approach^[1] - leverage **floorplanning** to:
 - Determine the maximum number of SST that can be instantiated
 - Optimize SSTs placement to ease timing closure

[1] Rabozzi, M., Natale, G., Festa, B., Miele, A., & Santambrogio, M. D. (2017, September). Optimizing streaming stencil time-step designs via FPGA floorplanning. In Field Programmable Logic and Applications (FPL), 2017 27th International Conference on (pp. 1-4).

Experimental result



With floorplan

SSTs

90 (+2.3%)

Frequency

228 MHz (+10.7%)

Design time

~25 hours (16x less)

Without floorplan

SSTs

88

Frequency

206 MHz

Design time

~400 hours

Algorithm: **Jacobi2D**

Target FPGA: **Xilinx Virtex xc7vx485**



- **SST (Single Streaming Timestep)**
 - Scientific applications with a regular execution flow in which
 - a N-dimensional data structure is updated over time
 - Example: Heat flow simulation, Gauss-Seidel linear systems solver
- **Dataflow**
 - Applications with a static compute graph where the same computation is repeated for multiple input items
 - Example: Asian Option Pricing, Image processing filters
- **Master / Slave**
 - More general applications with a regular execution flow, ideal for working set that can be effectively tiled
 - Example: N-Body Physics Simulation, Retinal Vessel Segmentation



```
void foo(type_1* in_1, type_2* in_2, scalar_type_1* v1, type_1* out_1){
```

```
    for(int i = offs; i < l_SIZE; i++){
```

```
        S1: ...statements...
```

```
        for(int j = ...; j < 15; j++){
```

```
            S2: ...statements...
```

```
        }
```

```
        S3: ...statements...
```

```
        for(int j = ... ){
```

```
            S4: ...statements...
```

```
        }
```

```
}
```

```
    for(int i = offs_3; ... ){
```

```
        S5: ...statements...
```

```
}
```

```
}
```

OUTER STREAMING LOOPS

```
void foo(type_1* in_1, type_2* in_2, scalar_type_1* v1, type_1* out_1){
```

```
    for(int i = offs; i < l_SIZE; i++){
```

```
        S1: ...statements...
```

```
        for(int j = ...; j < 15; j++){
```

```
            S2: ...statements...
```

```
}
```

```
        S3: ...statements...
```

```
        for(int j = ... ){
```

```
            S4: ...statements...
```

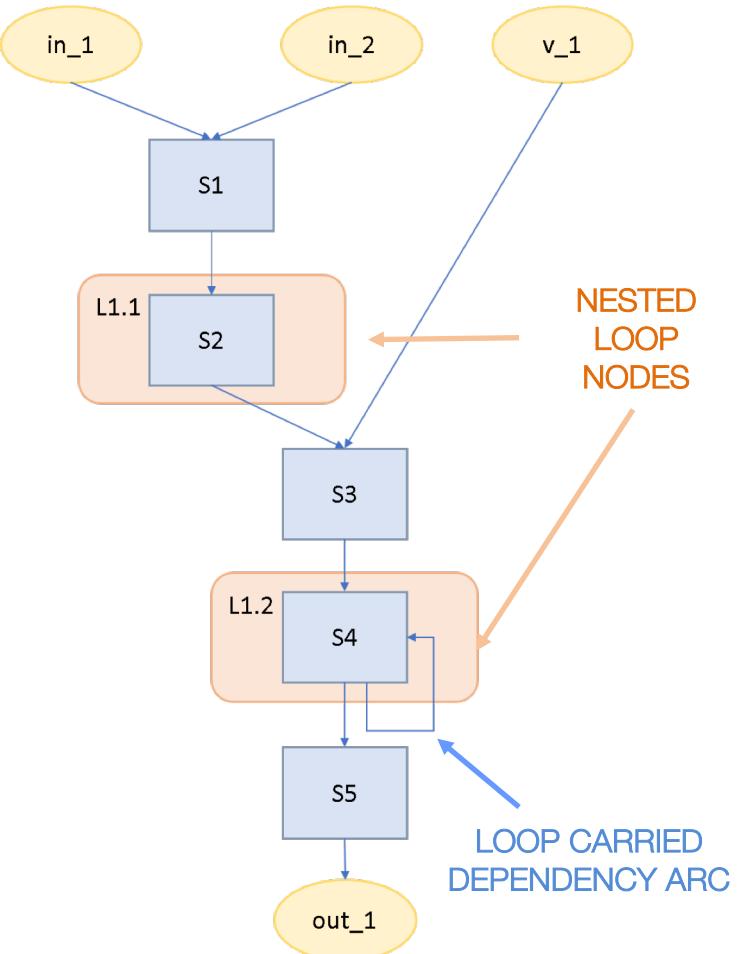
```
}
```

```
    for(int i = offs_3; ... ){
```

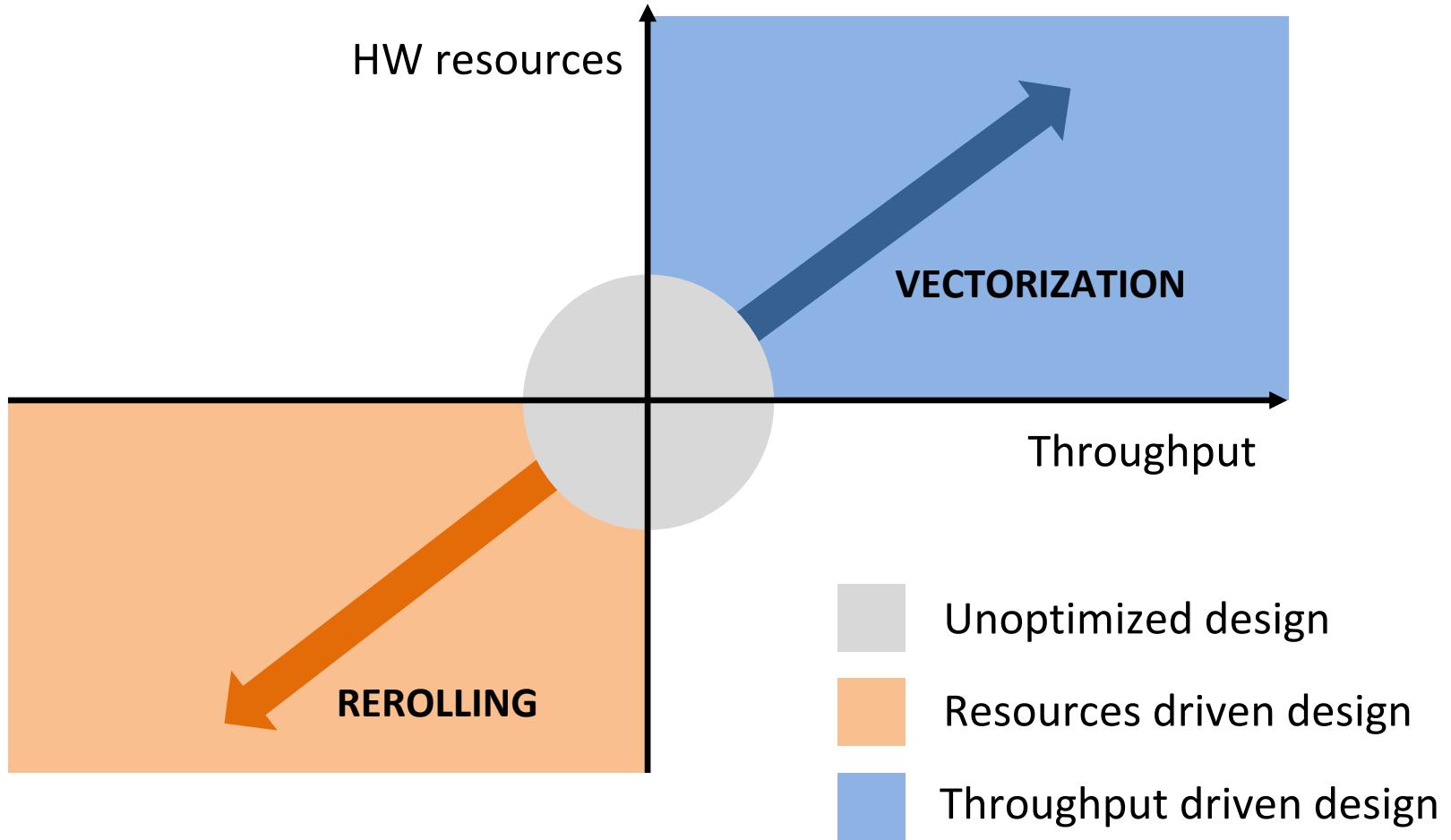
```
        S5: ...statements...
```

```
}
```

OUTER STREAMING LOOPS



Design optimizations



REROLLING:

- Nested loops are unrolled by default
- Rerolling can be applied only to loops without carried dependencies



Which optimization to apply?



We need:

- Resource estimation model
- Performance estimation model



We need:

- Resource estimation model
- Performance estimation model



Tailored for each
optimization





Rerol. Factor	Speedup	DSP/LUT Balance
30	15.83	1.0
15	31.22	1.0
10	45.95	0.1
8	56.91	1.0
6	74.35	0.1
5	88.10	0.1

Total Resources			
DSP	BRAM	FF	LUT
39.06	37.65	21.86	37.15
55.08	42.09	24.47	40.71
29.30	41.50	31.55	52.32
87.11	54.30	29.60	47.73
41.02	59.13	39.51	64.31
46.88	63.96	43.31	70.09

- System implemented with MaxCompiler on a **Galava MAX4**
- Baseline software implementation executed on an **Intel(R) Core(TM) i7-6700** CPU at 3.40GHz, compiled with **gcc 4.4.7** and -O3 optimization

Peverelli F., Rabozzi M., Del Sozzo E., & Santambrogio, M.D. (2018, May). OXiGen: A tool for automatic acceleration of C functions into dataflow FPGA-based kernels. In Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2018 IEEE International.

Testbench parameters:

- Averaging points window: 30
- Asian Options in portfolio: 10000
- Market scenarios: 5000

Execution time	
CAOS [1]	Hand-tuned [2]
1.78s	1.25s

[1] Peverelli F., Rabozzi M., Del Sozzo E., & Santambrogio, M.D. (2018, May). OXiGen: A tool for automatic acceleration of C functions into dataflow FPGA-based kernels. In Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2018 IEEE International.

[2] Nestorov, A. M., Reggiani, E., Palikareva, H., Burovskiy, P., Becker, T., & Santambrogio, M. D. (2017, May). A Scalable Dataflow Implementation of Curran's Approximation Algorithm. In Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International (pp. 150-157). IEEE.

CAOS vs Hand-tuned implementation

Testbench parameters:

- Averaging points window: 30
- Asian Options in portfolio: 10000
- Market scenarios: 5000

Execution time	
CAOS [1]	Hand-tuned [2]
1.78s	1.25s

About a day of work vs several weeks!

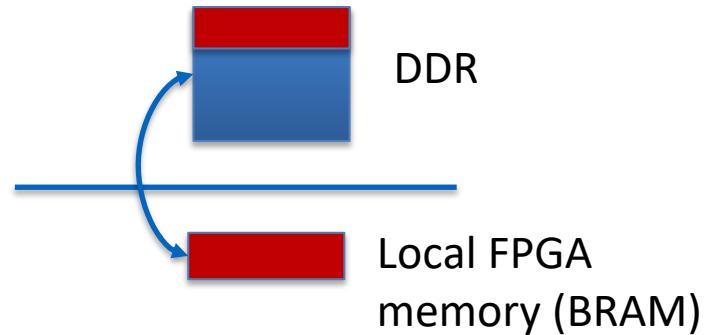
[1] Peverelli F., Rabozzi M., Del Sozzo E., & Santambrogio, M.D. (2018, May). OXiGen: A tool for automatic acceleration of C functions into dataflow FPGA-based kernels. In Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2018 IEEE International.

[2] Nestorov, A. M., Reggiani, E., Palikareva, H., Burovskiy, P., Becker, T., & Santambrogio, M. D. (2017, May). A Scalable Dataflow Implementation of Curran's Approximation Algorithm. In Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International (pp. 150-157). IEEE.



- **SST (Single Streaming Timestep)**
 - Scientific applications with a regular execution flow in which
 - a N-dimensional data structure is updated over time
 - Example: Heat flow simulation, Gauss-Seidel linear systems solver
- **Dataflow**
 - Applications with a static compute graph where the same computation is repeated for multiple input items
 - Example: Asian Option Pricing, Image processing filters
- **Master / Slave**
 - More general applications with a regular execution flow, ideal for working set that can be effectively tiled
 - Example: N-Body Physics Simulation, Retinal Vessel Segmentation

- Support a larger class of source codes, ideally, close to the same set of codes supported by High Level Synthesis tools (e.g. Vivado HLS)
- Tiled computation:
 - Application's memory transferred in blocks from DDR to local FPGA memory
 - Computation performed on a block of data at a time





- Current Implementation
 - Relies on Vivado HLS for resource and performance estimation
 - Explore potential optimizations by testing HLS directives (e.g. #pragma HLS UNROLL, #pragma HLS PIPELINE, ...)
 - Quite accurate but slow (HLS synthesis time may be high)
- On going work:
 - Leverage and extend the classic **roofline model** for CPU to predict performance and potential optimizations
 - Potentially less accurate but faster, allowing to explore a larger optimization space

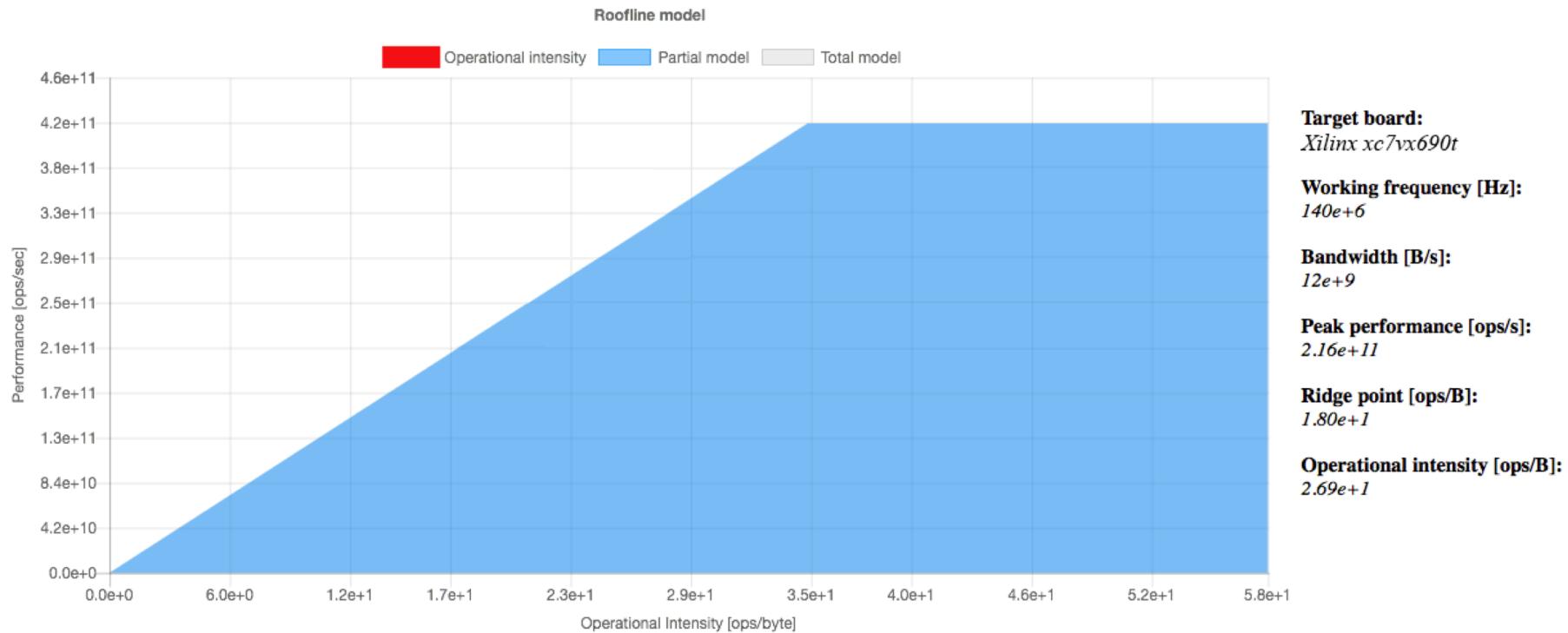


PROBLEM:

- Theoretical peak performance dependent on the **types of operations** performed, their **bitwidth** and **technology mapping**

IDEA:

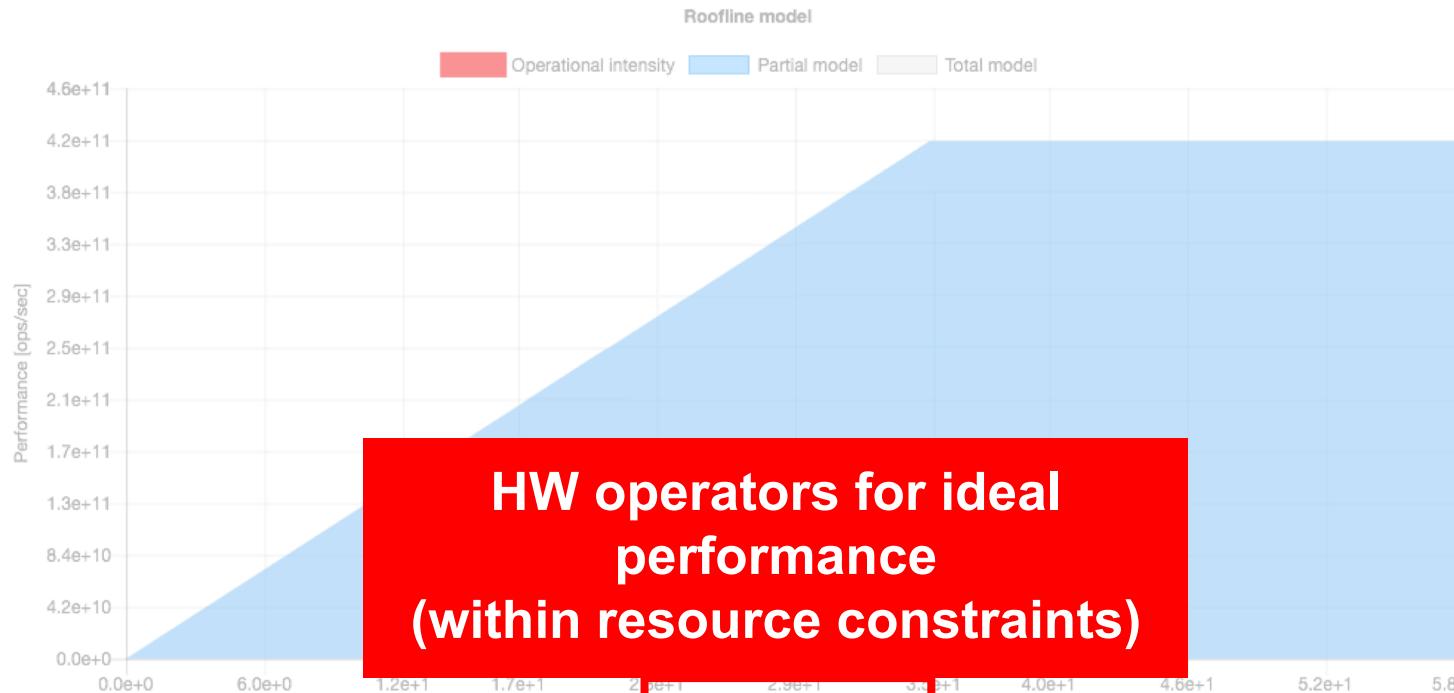
- Estimate peak performance for each combination of:
 - Target device
 - Operation type
 - Operands bitwidth
 - Target frequency
- Analyze the source code and derive a unified roofline according to the **operations count** within the code



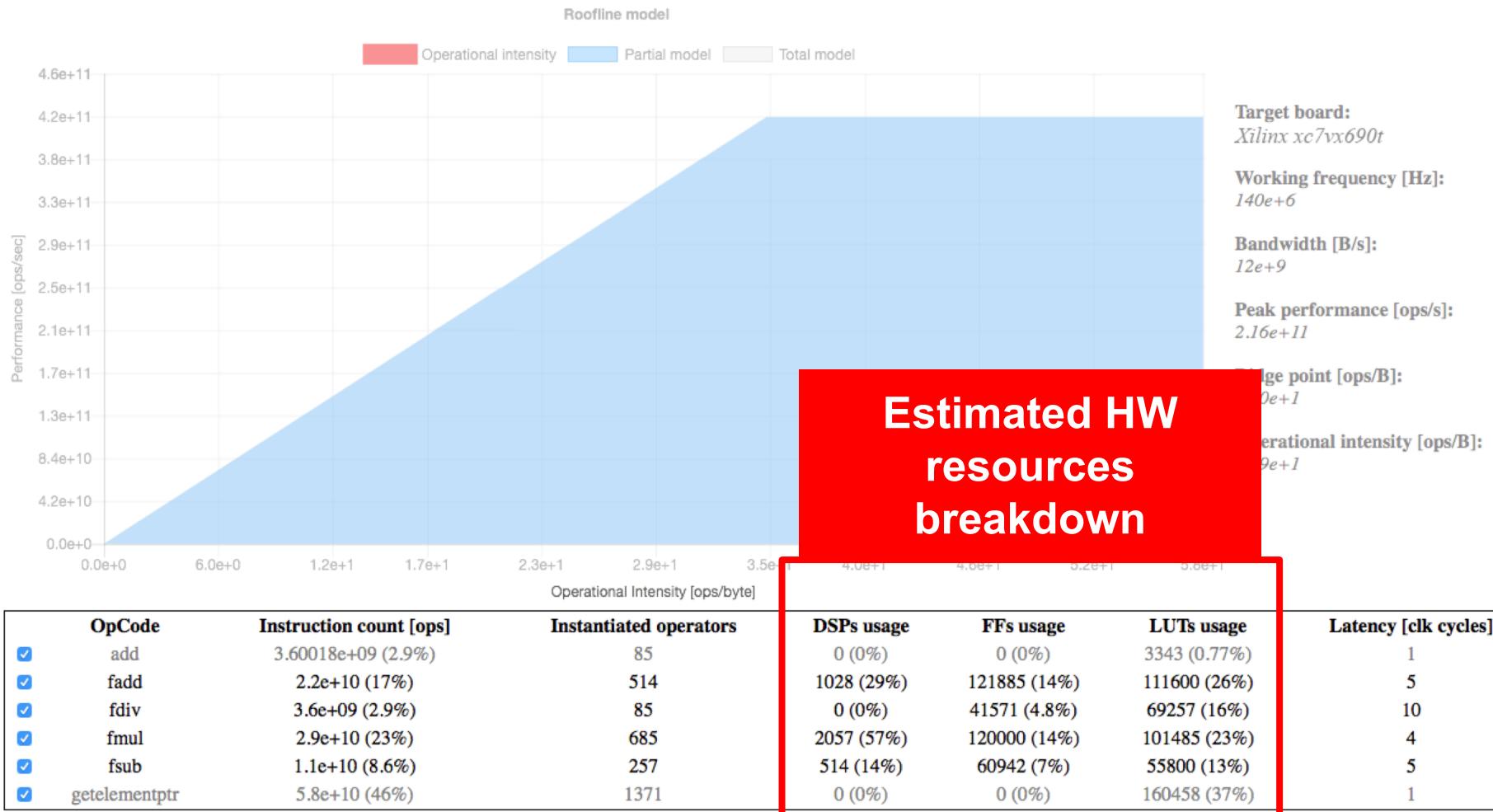
OpCode	Instruction count [ops]	Instantiated operators	DSPs usage	FFs usage	LUTs usage	Latency [clk cycles]
<input checked="" type="checkbox"/> add	$3.60018e+09$ (2.9%)	85	0 (0%)	0 (0%)	3343 (0.77%)	1
<input checked="" type="checkbox"/> fadd	$2.2e+10$ (17%)	514	1028 (29%)	121885 (14%)	111600 (26%)	5
<input checked="" type="checkbox"/> fdiv	$3.6e+09$ (2.9%)	85	0 (0%)	41571 (4.8%)	69257 (16%)	10
<input checked="" type="checkbox"/> fmul	$2.9e+10$ (23%)	685	2057 (57%)	120000 (14%)	101485 (23%)	4
<input checked="" type="checkbox"/> fsub	$1.1e+10$ (8.6%)	257	514 (14%)	60942 (7%)	55800 (13%)	5
<input checked="" type="checkbox"/> getelementptr	$5.8e+10$ (46%)	1371	0 (0%)	0 (0%)	160458 (37%)	1

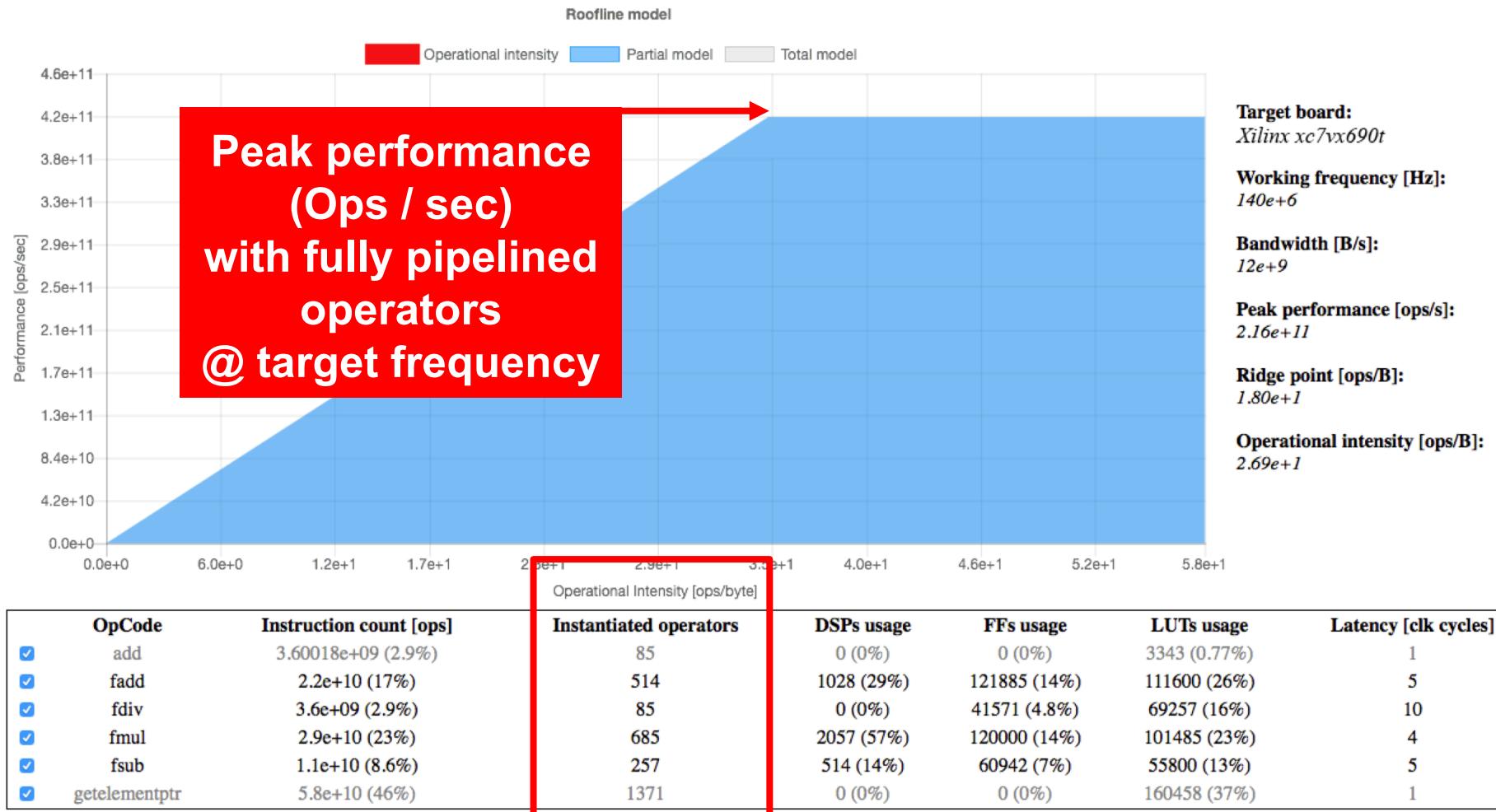


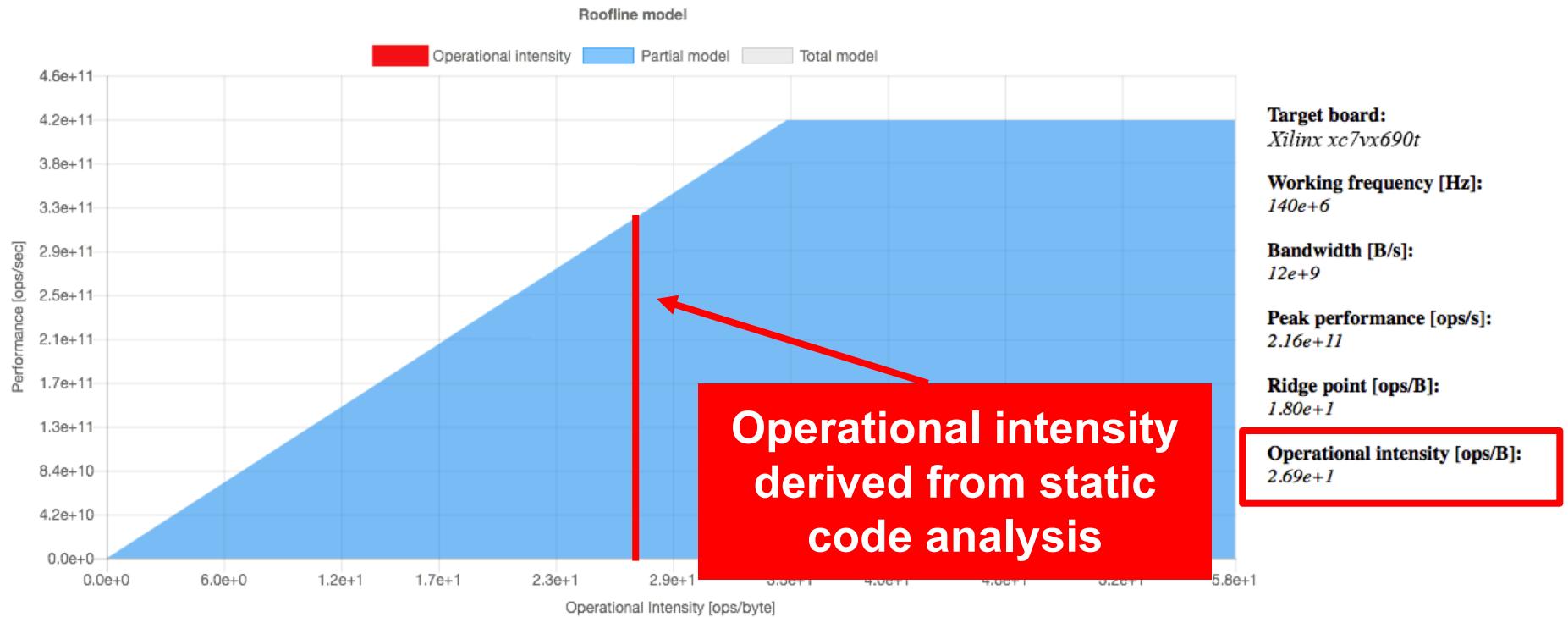
OpCode	Instruction count [ops]	Instantiated operators	DSPs usage	FFs usage	LUTs usage	Latency [clk cycles]
<input checked="" type="checkbox"/> add	3.60018e+09 (2.9%)	85	0 (0%)	0 (0%)	3343 (0.77%)	1
<input checked="" type="checkbox"/> fadd	2.2e+10 (17%)	514	1028 (29%)	121885 (14%)	111600 (26%)	5
<input checked="" type="checkbox"/> fdiv	3.6e+09 (2.9%)	85	0 (0%)	41571 (4.8%)	69257 (16%)	10
<input checked="" type="checkbox"/> fmul	2.9e+10 (23%)	685	2057 (57%)	120000 (14%)	101485 (23%)	4
<input checked="" type="checkbox"/> fsub	1.1e+10 (8.6%)	257	514 (14%)	60942 (7%)	55800 (13%)	5
<input checked="" type="checkbox"/> getelementptr	5.8e+10 (46%)	1371	0 (0%)	0 (0%)	160458 (37%)	1



OpCode	Instruction count [ops]	Instantiated operators	DSPs usage	FFs usage	LUTs usage	Latency [clk cycles]
<input checked="" type="checkbox"/> add	$3.60018e+09$ (2.9%)	85	0 (0%)	0 (0%)	3343 (0.77%)	1
<input checked="" type="checkbox"/> fadd	$2.2e+10$ (17%)	514	1028 (29%)	121885 (14%)	111600 (26%)	5
<input checked="" type="checkbox"/> fdiv	$3.6e+09$ (2.9%)	85	0 (0%)	41571 (4.8%)	69257 (16%)	10
<input checked="" type="checkbox"/> fmul	$2.9e+10$ (23%)	685	2057 (57%)	120000 (14%)	101485 (23%)	4
<input checked="" type="checkbox"/> fsub	$1.1e+10$ (8.6%)	257	514 (14%)	60942 (7%)	55800 (13%)	5
<input checked="" type="checkbox"/> getelementptr	$5.8e+10$ (46%)	1371	0 (0%)	0 (0%)	160458 (37%)	1







OpCode	Instruction count [ops]	Instantiated operators	DSPs usage	FFs usage	LUTs usage	Latency [clk cycles]
<input checked="" type="checkbox"/> add	3.60018e+09 (2.9%)	85	0 (0%)	0 (0%)	3343 (0.77%)	1
<input checked="" type="checkbox"/> fadd	2.2e+10 (17%)	514	1028 (29%)	121885 (14%)	111600 (26%)	5
<input checked="" type="checkbox"/> fdiv	3.6e+09 (2.9%)	85	0 (0%)	41571 (4.8%)	69257 (16%)	10
<input checked="" type="checkbox"/> fmul	2.9e+10 (23%)	685	2057 (57%)	120000 (14%)	101485 (23%)	4
<input checked="" type="checkbox"/> fsub	1.1e+10 (8.6%)	257	514 (14%)	60942 (7%)	55800 (13%)	5
<input checked="" type="checkbox"/> getelementptr	5.8e+10 (46%)	1371	0 (0%)	0 (0%)	160458 (37%)	1



We developed a GitHub repository with 4 example applications (https://github.com/necst/CAOS_examples) targeting AWS F1 instances:

- Vector Addition
- Variational Monte Carlo
- Retinal Vessel Segmentation
- Smith-Waterman