2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
10 YEAR ANNIVERSARY

MapGL: Interactive Application Mapping and Profiling on a Grid of Processing Cells

Claudio Raccomandato
Politecnico di Torino

Emad M. Arasteh
Fowler School of Engineering
Chapman University

Rainer Dömer
University of California, Irvine

The participation to DVCon Europe 2023 has been supported by NXP Semiconductors

accellera
SYSTEMS INITIATIVE

Outline

- **Introduction**
  - Multi-core processors memory bottleneck
  - Grid of Processing Cells (GPC)
  - The mapping problem
- **Map Grid-based Layouts (MapGL)**
  - Overview
  - Modules
  - Channels and MARI library
  - Performance exploration
  - Project export
- **Experiments and Results**
  - GoogLeNet CNN
  - GoogLeNet on GPC
    - MapGL demo
  - GoogLeNet analysis
    - Memory usage
    - FIFOs delay
    - Cores delay
- **Conclusion**
  - Future Work
  - Acknowledgments

Multi-core Processors Memory Bottleneck

Current architectures

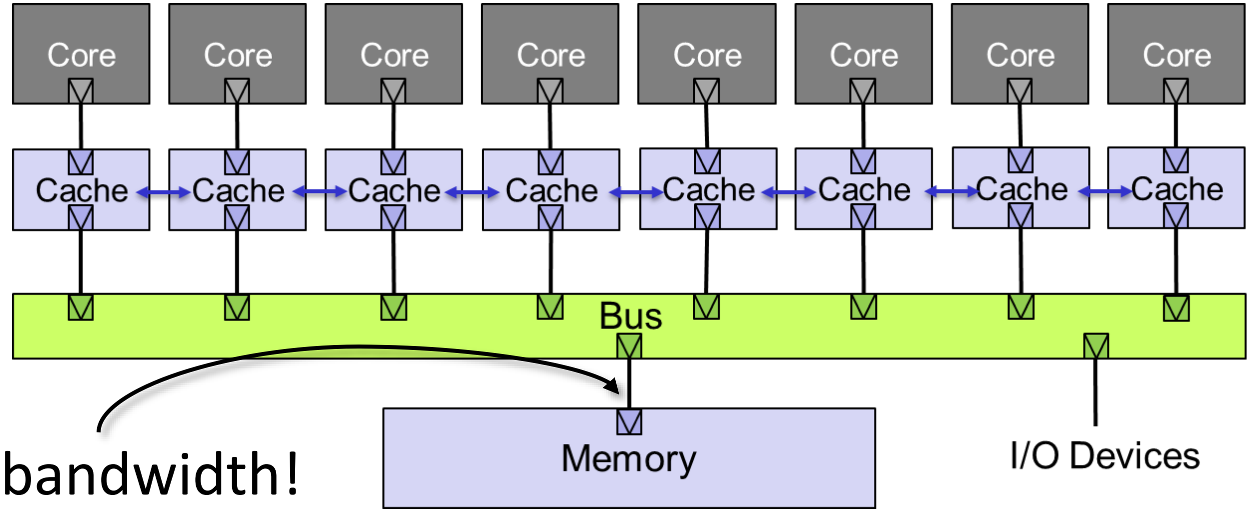
- Shared interconnect and single memory [1]

Limitations

- By scaling the number of processors
  - Memory bandwidth increases
  - Interconnect contention grows
  - Execution time slows down
  - Caches become necessary

Observations

➢ Cache coherence is complicated and expensive

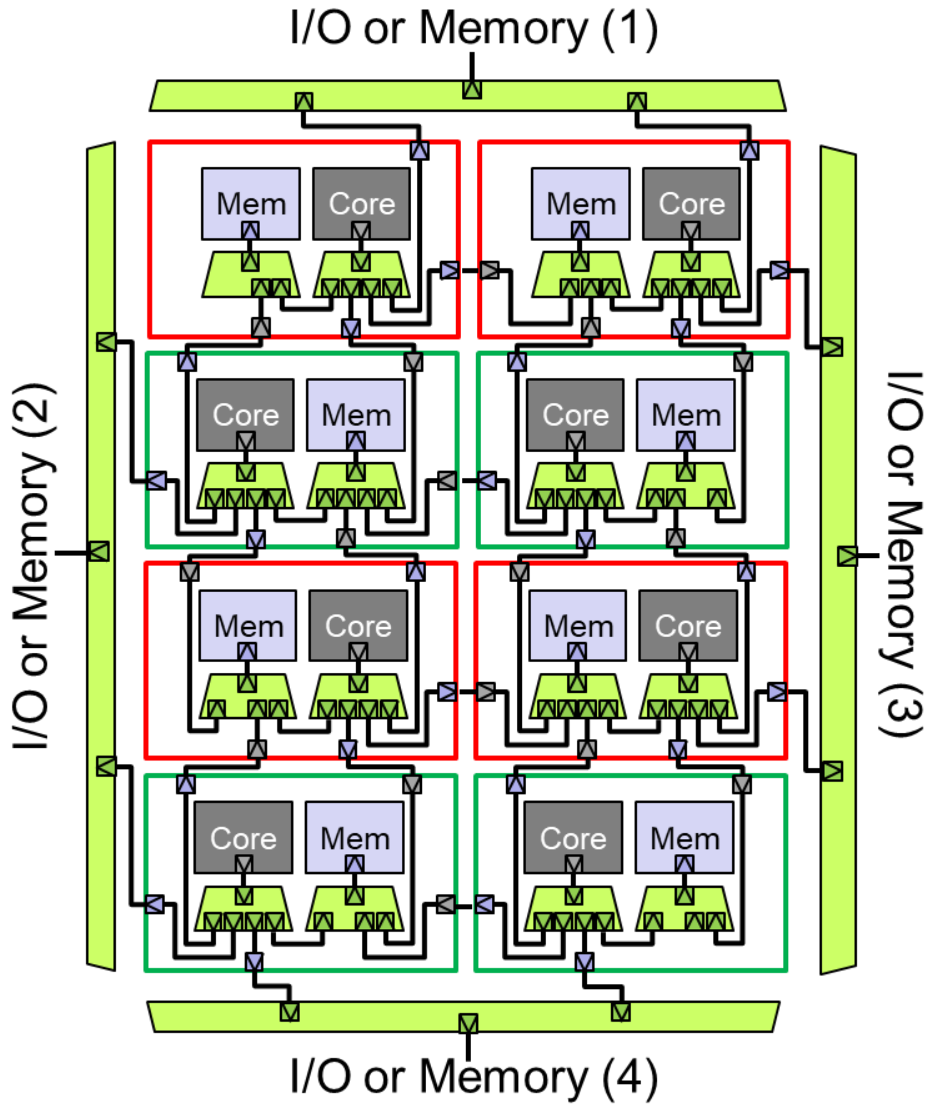


High bandwidth!

Grid of Processing Cells (GPC)

Alternative approach

- GPC architecture [2]
    - Distributed memories
    - Less contention, max 4 cores connected to each memory
    - Scalable along 2 dimensions
    - ➢ Cache-less



Limitations

- **Software partitioning** increases in complexity
- **Programmability** of grid platforms is not trivial
- Higher number of **cores interactions**
- No standard way of **evaluating performance**

The mapping problem

Manually writing functions

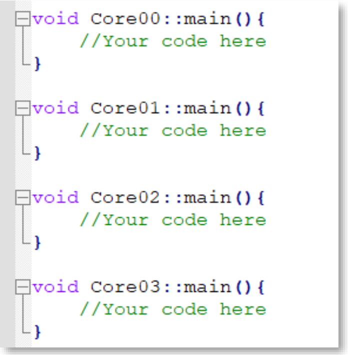
- Error-prone with more than 10 cores
- Tedious without visual information, especially for core communication

Spreadsheet approach

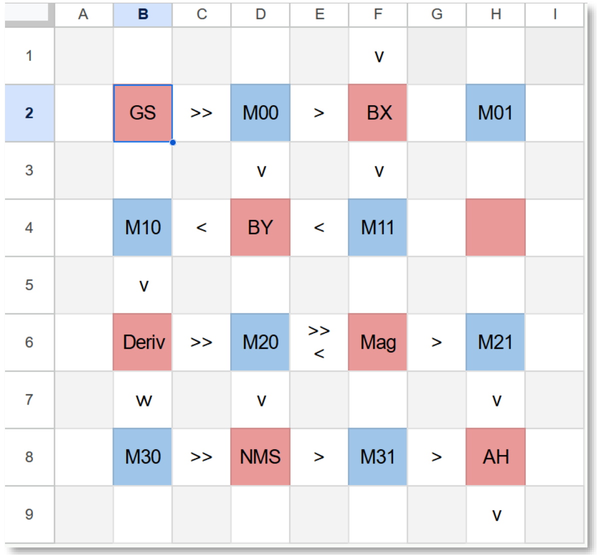
- Fast prototyping
- Clear inner core communication
- Auto-generated code

Limitations

- Channels parameters manually set in the source code



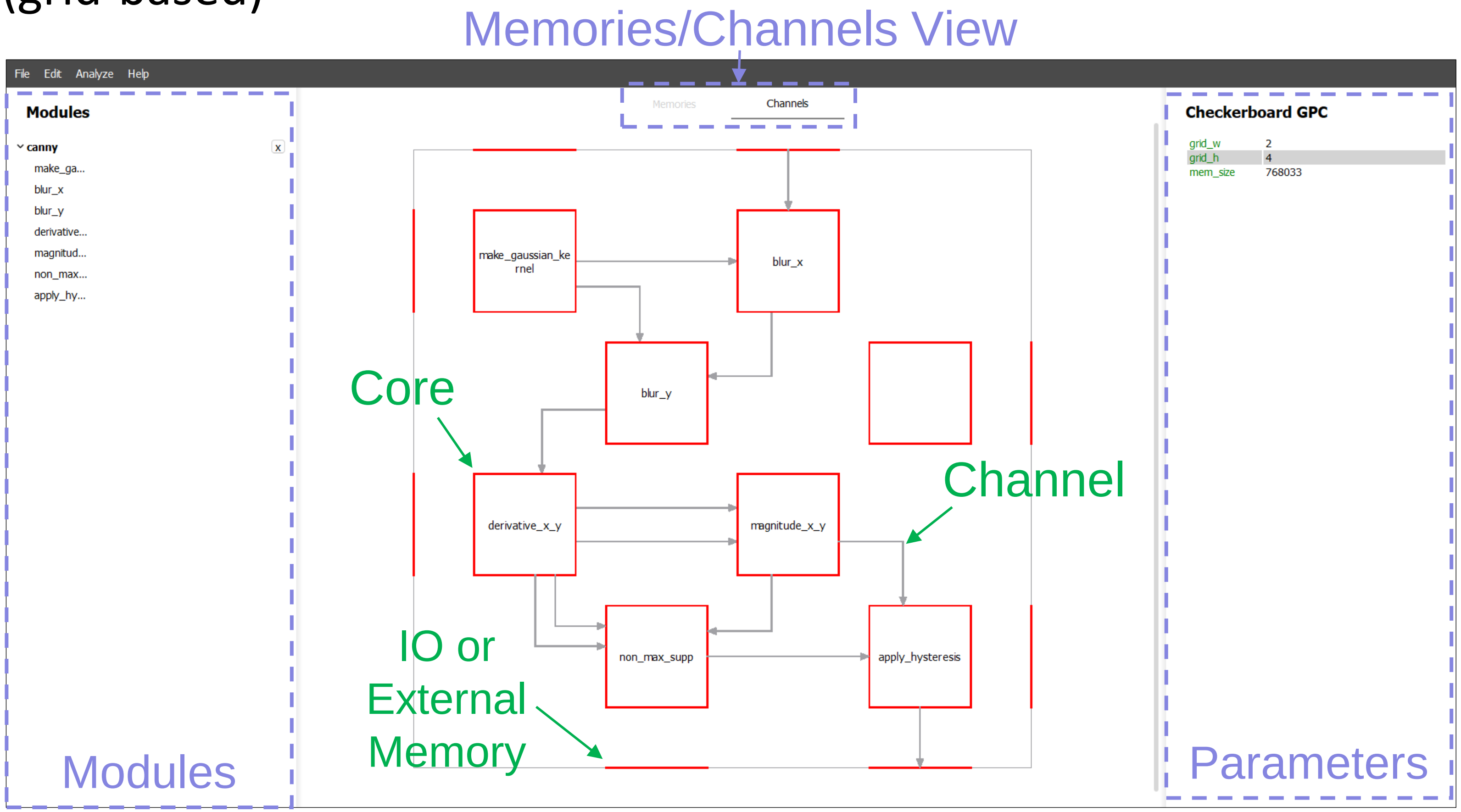
```
void Core00::main(){
    //Your code here
}

void Core01::main(){
    //Your code here
}

void Core02::main(){
    //Your code here
}

void Core03::main(){
    //Your code here
}
```



	A	B	C	D	E	F	G	H	I
1						v			
2		GS	>>	M00	>	BX		M01	
3				v		v			
4		M10	<	BY	<	M11			
5		v							
6		Deriv	>>	M20	>> <	Mag	>	M21	
7		w		v				v	
8		M30	>>	NMS	>	M31	>	AH	
9								v	

Our proposal: MapGL

- CAD software that generates SystemC models and evaluate performances of GPC-like processors (grid-based)



Modules

- canny
  - make_ga...
  - blur_x
  - blur_y
  - derivative...
  - magnitud...
  - non_max...
  - apply_hy...

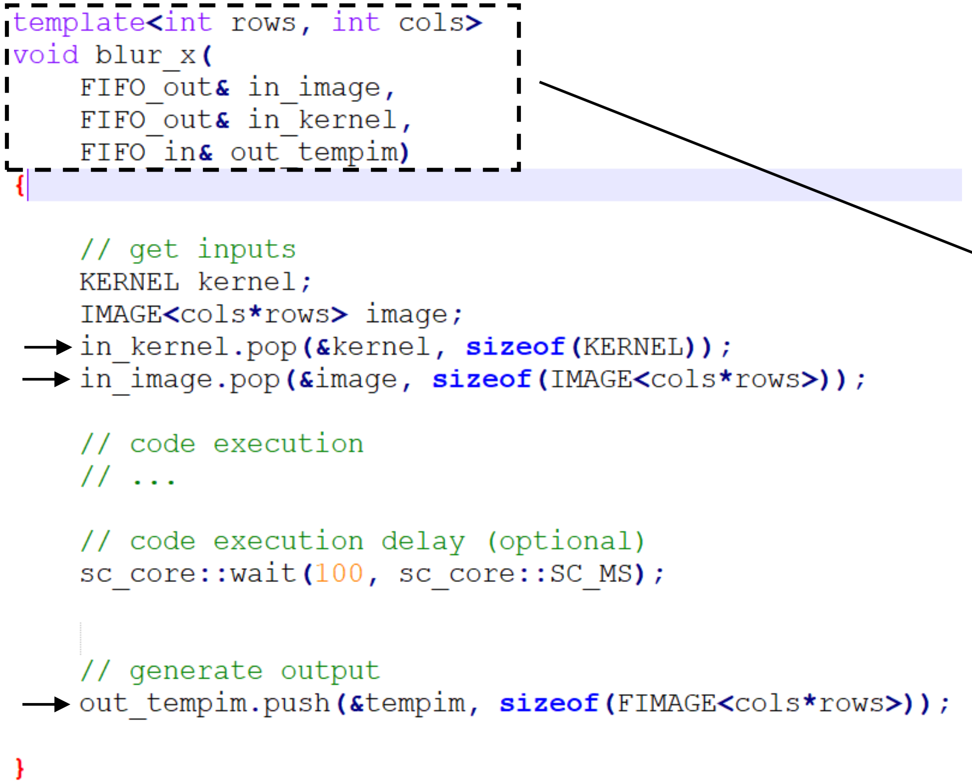
Checkerboard GPC

grid_w	2
grid_h	4
mem_size	768033

Modules

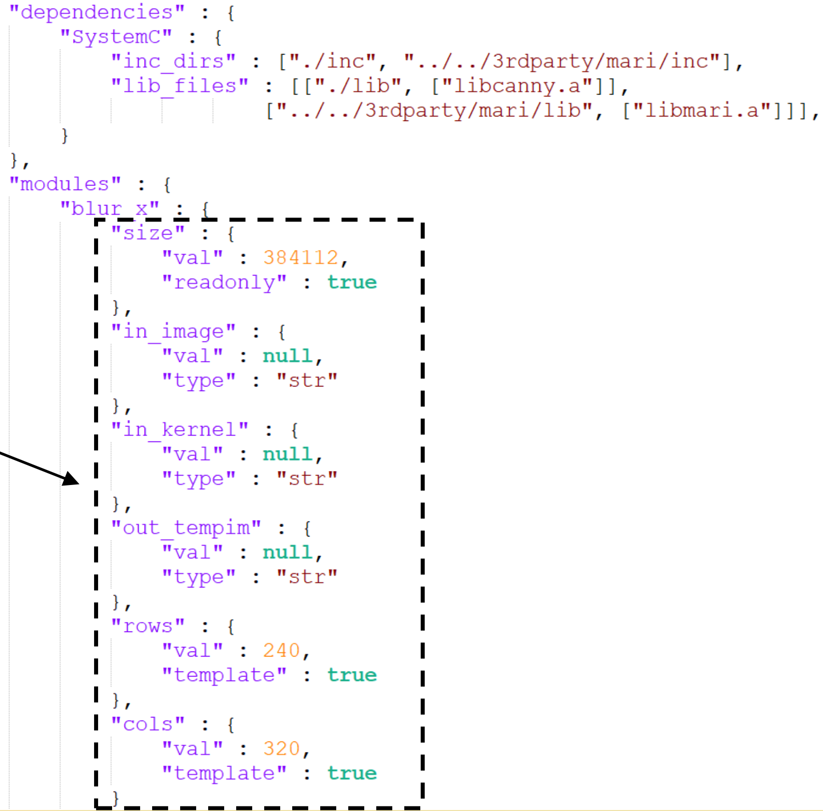
- ➢**Module**: describes the behavior of a core using C/C++ code
- Custom Modules can be imported in a project using a user defined JSON file

blur_x.hpp



```
template<int rows, int cols>
void blur_x(
    FIFO_out& in_image,
    FIFO_out& in_kernel,
    FIFO_in& out_tempim)
{

    // get inputs
    KERNEL kernel;
    IMAGE<cols*rows> image;
→   in_kernel.pop(&kernel, sizeof(KERNEL));
→   in_image.pop(&image, sizeof(IMAGE<cols*rows>));

    // code execution
    // ...

    // code execution delay (optional)
    sc_core::wait(100, sc_core::SC_MS);


    // generate output
→   out_tempim.push(&tempim, sizeof(FIMAGE<cols*rows>));

}
```

blur_x.json (imported in MapGL)

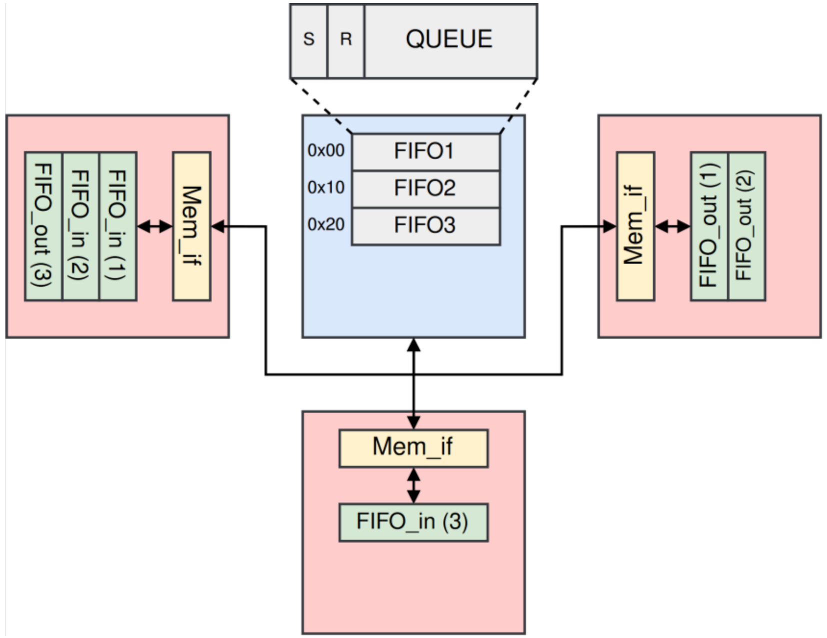


```
"dependencies" : {
    "SystemC" : {
        "inc_dirs" : ["./inc", "../../3rdparty/mari/inc"],
        "lib_files" : [["./lib", ["libcanny.a"]],
                       ["../../3rdparty/mari/lib", ["libmari.a"]]],
    }
},
"modules" : {
    "blur_x" : {
        "size" : {
            "val" : 384112,
            "readonly" : true
        },
        "in_image" : {
            "val" : null,
            "type" : "str"
        },
        "in_kernel" : {
            "val" : null,
            "type" : "str"
        },
        "out_tempim" : {
            "val" : null,
            "type" : "str"
        },
        "rows" : {
            "val" : 240,
            "template" : true
        },
        "cols" : {
            "val" : 320,
            "template" : true
        }
```

Channels and MARI library

➢ **Channel**: allows two or more cores to communicate

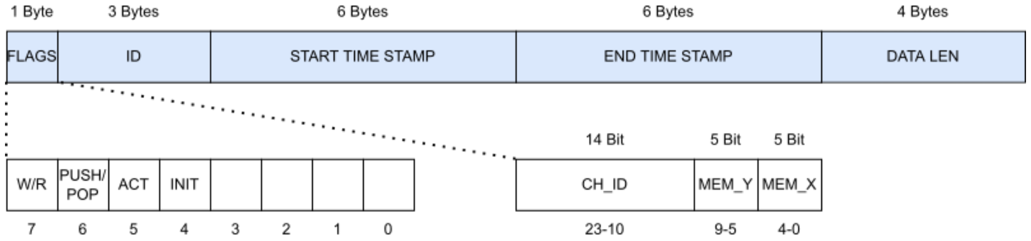
- MapGL uses channels from our MARI library



MARI contains two type of interfaces:

1. **Memory interface**, connects a core to one memory through TLM-2.0 Generic Protocol and blocking communication
2. **Channel interface**, create a software channel in the memory (e.g. FIFO) which enable cores communication using methods like *push()* and *pop()*

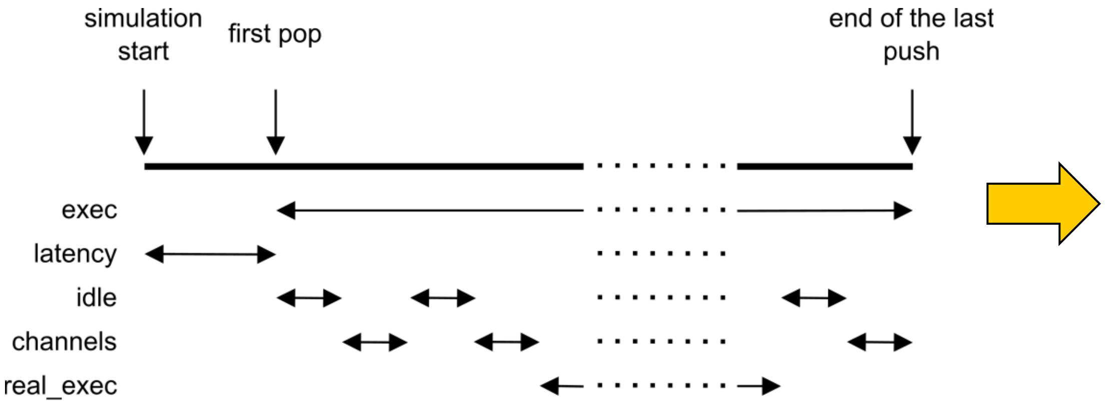
- For debugging purposes, MARI can generate a record of the memory accesses
- Memory accesses are encoded in a binary file

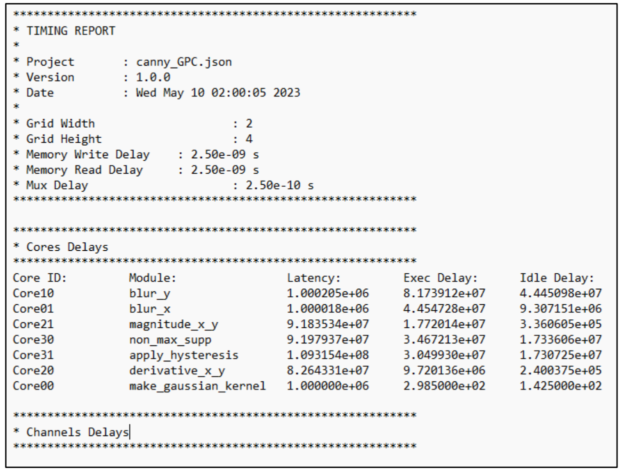


Performance exploration

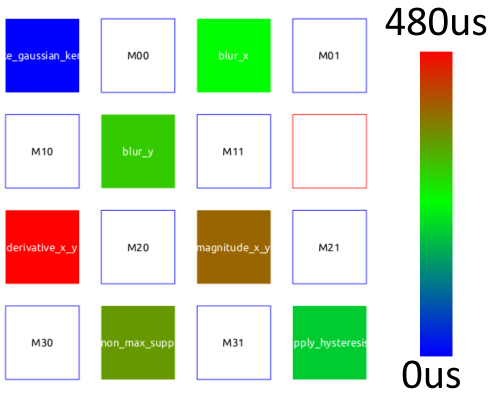
MapGL can profile the architecture with two types of analysis:

- **Memories usage analysis**, statically evaluates how much space each memory uses for storing application and channels. Then it generates a detailed report and a heatmap
- **Timing analysis**, fine-grain architecture latencies evaluation:
    - MapGL uses the memory accesses recorded with the MARI library to backtrack the application timing and identify various delay contributions. Then it generates a detailed report and one heatmap per delay contribution





```
***********************************************************
* TIMING REPORT
*
* Project       : canny_GPC.json
* Version       : 1.0.0
* Date          : Wed May 10 02:00:05 2023
*
* Grid Width                    : 2
* Grid Height                   : 4
* Memory Write Delay    : 2.50e-09 s
* Memory Read Delay     : 2.50e-09 s
* Mux Delay                     : 2.50e-10 s
***********************************************************

***********************************************************
* Cores Delays
***********************************************************
Core ID:         Module:                Latency:         Exec Delay:      Idle Delay:
Core10           blur_y                 1.000205e+06     8.173912e+07     4.445098e+07
Core01           blur_x                 1.000018e+06     4.454728e+07     9.307151e+06
Core21           magnitude_x_y          9.183534e+07     1.772014e+07     3.360605e+05
Core30           non_max_supp           9.197937e+07     3.467213e+07     1.733606e+07
Core31           apply_hysteresis       1.093154e+08     3.049930e+07     1.730725e+07
Core20           derivative_x_y         8.264331e+07     9.720136e+06     2.400375e+05
Core00           make_gaussian_kernel   1.000000e+06     2.985000e+02     1.425000e+02

***********************************************************
* Channels Delays
***********************************************************
```

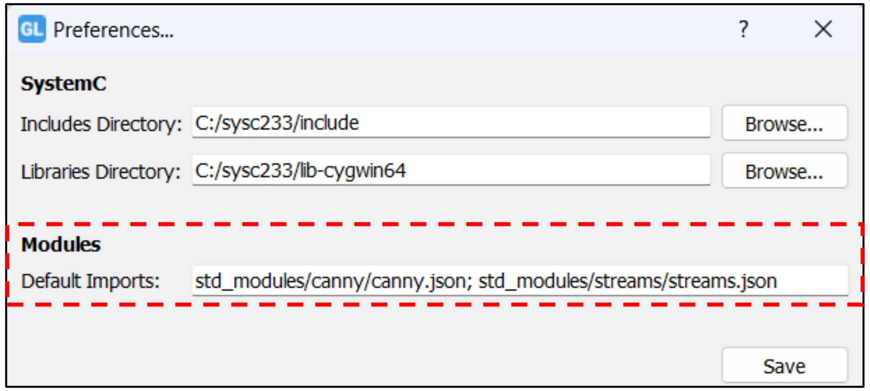


Project export

- MapGL project is stored inside a single JSON file
- During the export, custom modules dependencies get referenced

➢ The SystemC model get generated with an initial testbench

- The generated Makefile allows users to compile and run the simulation immediately



GL Preferences...

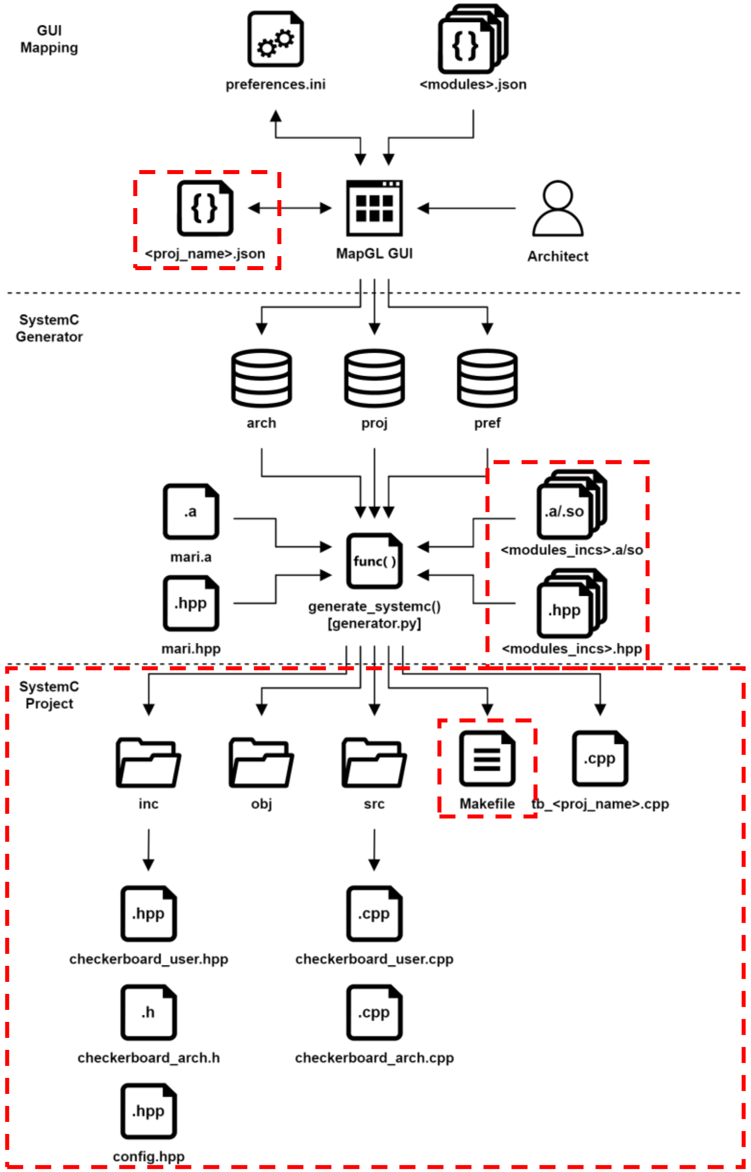
SystemC

Includes Directory: C:/sysc233/include

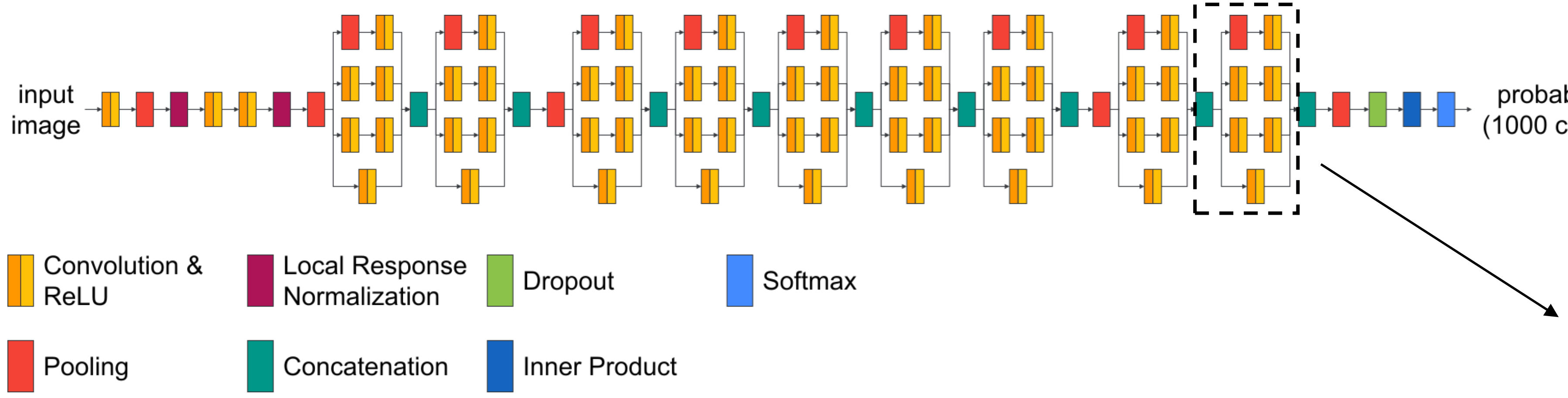
Libraries Directory: C:/sysc233/lib-cygwin64

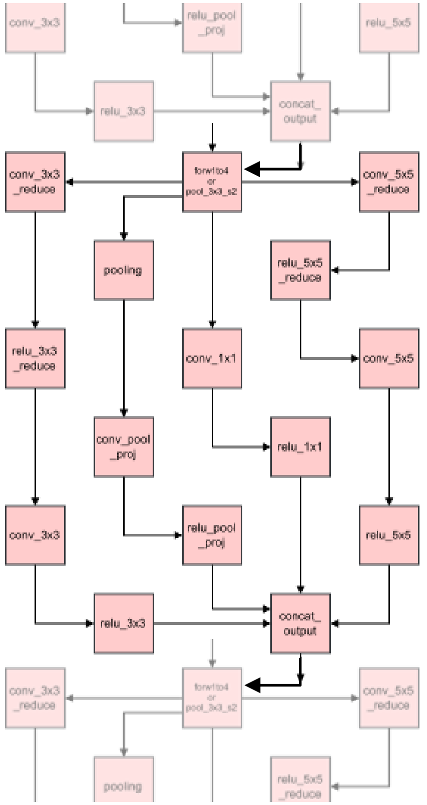
Modules

Default Imports: std_modules/canny/canny.json; std_modules/streams/streams.json



Case Study: GoogLeNet CNN





- The GoogLeNet is a state-of-the-art CNN for image classification composed of **142 layers** [3]

Mapping strategy

- 1 layer per core, exploiting GPC scalability
- Use as few additional core as possible to reduce area
- Map first the patterns that repeats the most and make them modular
    - Input and output on the same axis

GoogLeNet on GPC

Mapping

➢150 cores, 15x10 grid

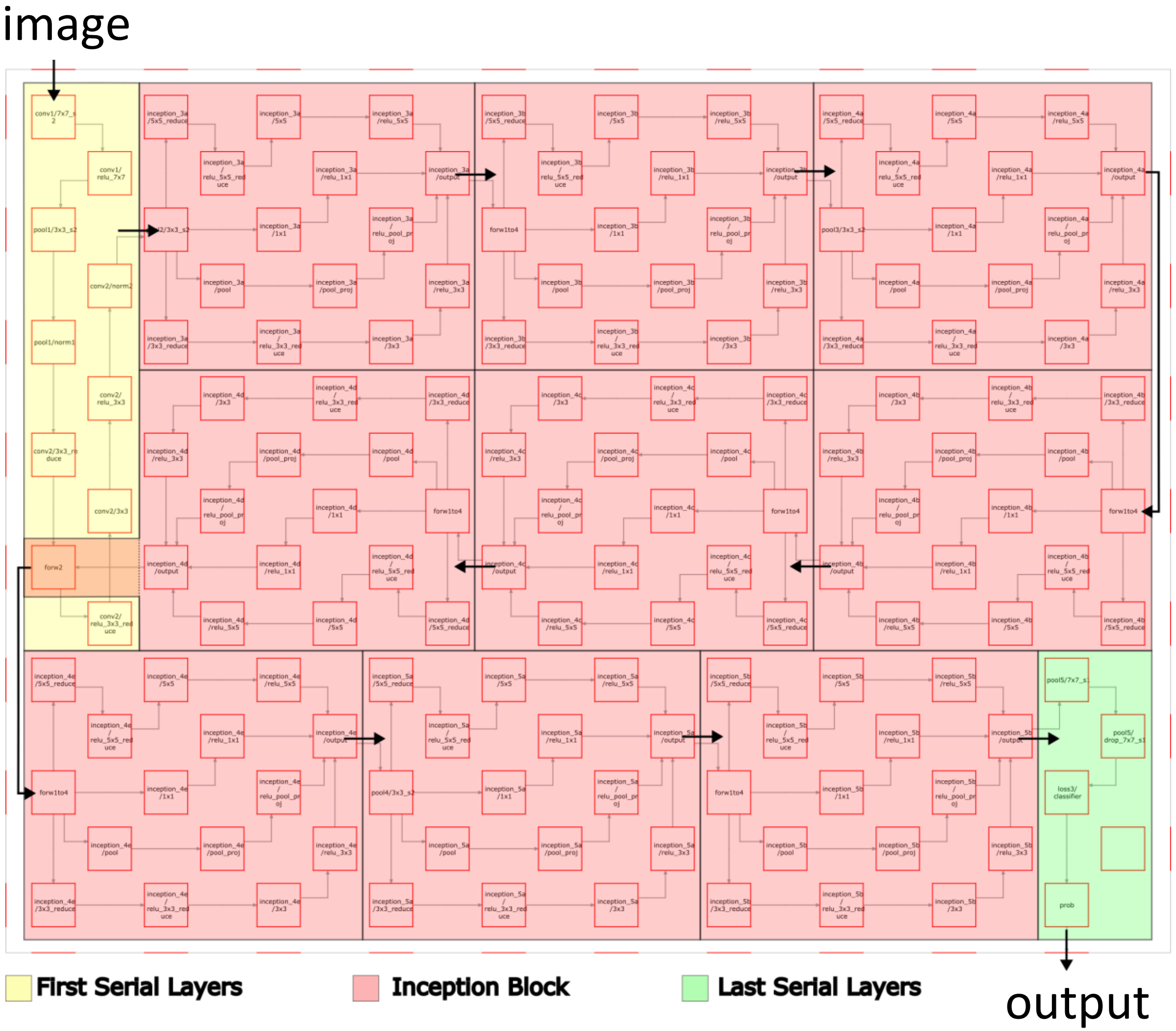
- 7 cores used for data forwarding
- 1 unused core
- Reuse of the same inception block mapping thanks to its modularity

High-speed model (<u>large FIFO</u>)

- FIFO size set equal to payload size
- less stalls → less delay

Low-memory model (<u>small FIFO</u>)

- FIFO size set to 64B



GoogLeNet memory usage

- Each memory can store up to 8MB
- Blue memories low memory usage
- Red memories high memory usage

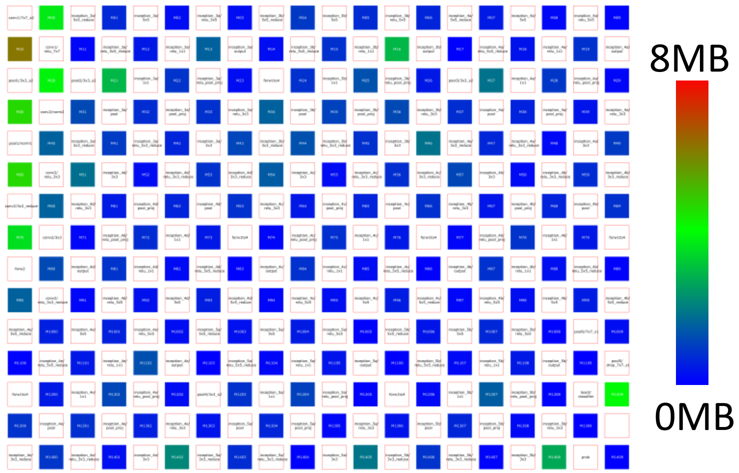
Observations

- The first transformations of the input image require more memory
- the smaller FIFOs used in the low-memory model flatten the memory usage

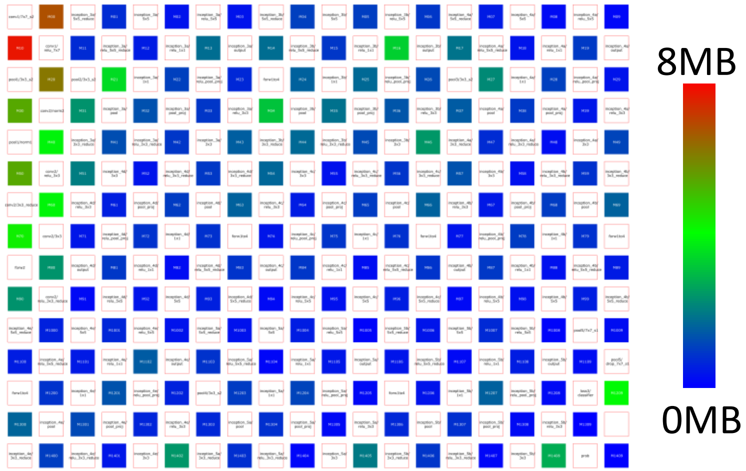
Total memory usage

- *high-speed*: **175MB** of which **52MB** are used by the FIFOs
- *low-memory*: **123MB** of which **13KB** are used by the FIFOs

low-memory



high-speed



GoogLeNet FIFOs delay

- MapGL normalizes the timing heatmaps, the red core represent the slowest

Setup

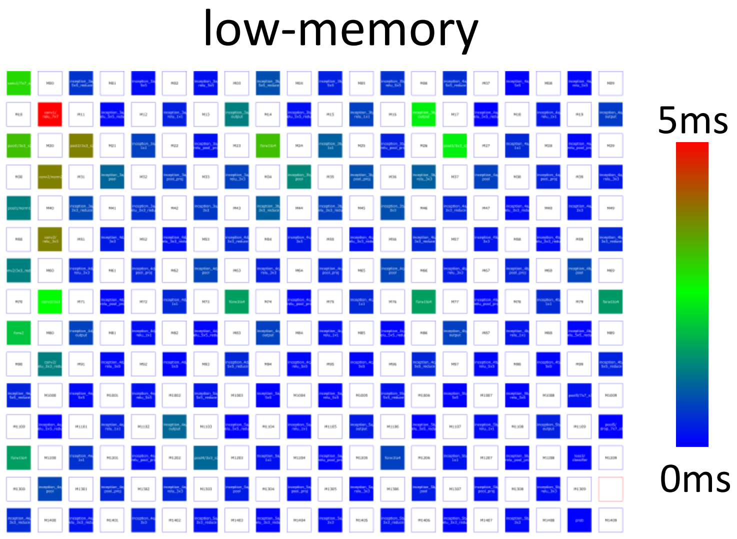
- Off-chip DRAM: 50ns read/write
- On-chip SRAM: 2.5ns read/write
- Multiplexer propagation: 0.25ns

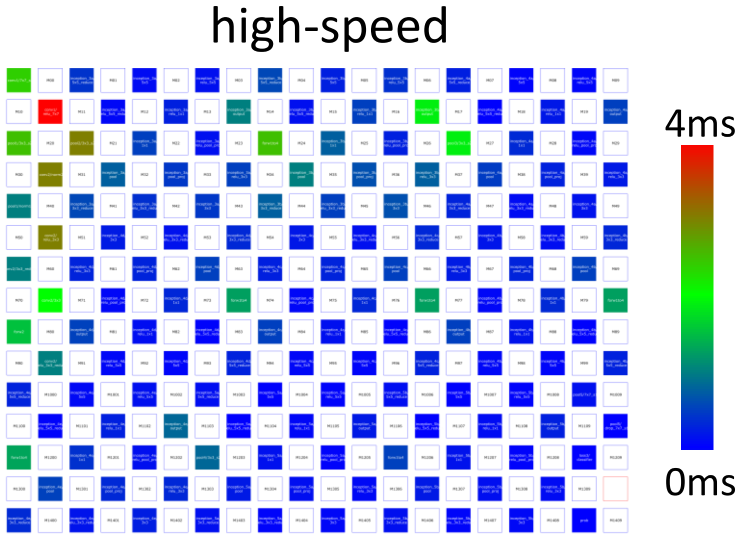
Observations

- The first transformations of the input image have higher communication delays
- Correlation between the payload size and its delay

Total FIFOs delay

- *high-speed*: **82ms**
- *low-memory*: **100ms**


low-memory


high-speed

GoogLeNet cores delay

- MapGL normalizes the timing heatmaps, the red core represent the slowest

Setup

- The computational delay of each core was calculated starting from the complexity of each layer (number of additions and multiplications)
- The Ara vector processor [4] was used as reference with its 16.9 DP-GFLOPS

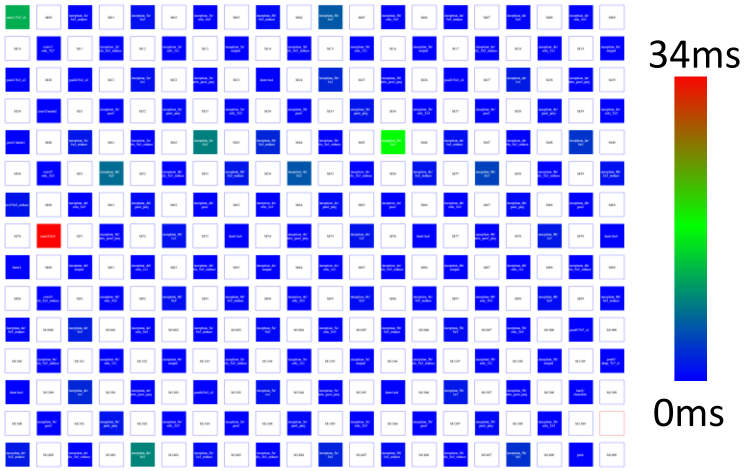
Observations

- The third convolution takes the most time

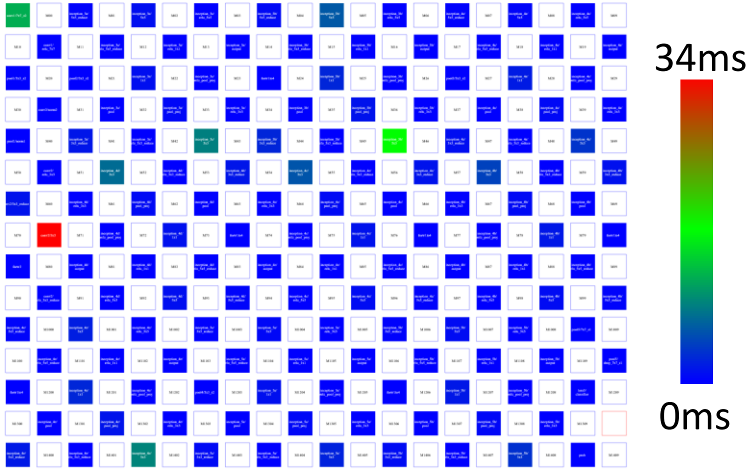
Total cores delay

- *high-speed / low-memory*: **117ms**
- Throughput feeding 500 images: **~23 fps**

low-memory



high-speed



Conclusion and Future Work

- MapGL editor was introduced
- A GPC-based GoogLeNet CNN was interactively mapped
- Two SystemC models were automatically generated and studied using MapGL built-in analysis tools and MARI library
- The results showed that the low-memory model is a valuable alternative to the high-speed one to highly reduce memory usage
- Hidden bottleneck was identified, a fine-grain pipeline would increase the throughput

Future work

- Automatically adjust the application mapping based on the results
- Support for other grid-base architectures
- Verilog/VHDL project export capability
- Integration with the GPC compiler

Acknowledgments

- Thanks to NXP Semiconductors for sponsoring the participation to DVCon Europe 2023
- And Politecnico di Torino for supporting the research

References

[1] G. Liu, T. Schmidt, A. Dingankar, D. Kirkpatrick, and R. Dömer, “Optimizing Thread-to-Core Mapping on Manycore Platforms with Distributed Tag Directories,” in Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC), Jan. 2015

[2] R. Dömer, “A Grid of Processing Cells (GPC) with Local Memories,” Center for Embedded and Cyber-physical Systems, University of California, Irvine, Tech. Rep. CECS-TR-22-01, Apr. 2022.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9

[4] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, “Ara: A 1-ghz+ scalable and energy-efficient risc-v vector processor with multiprecision floating-point support in 22-nm fd-soi,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 2, pp. 530–543, 2020