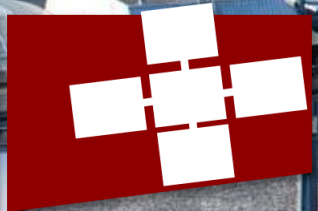


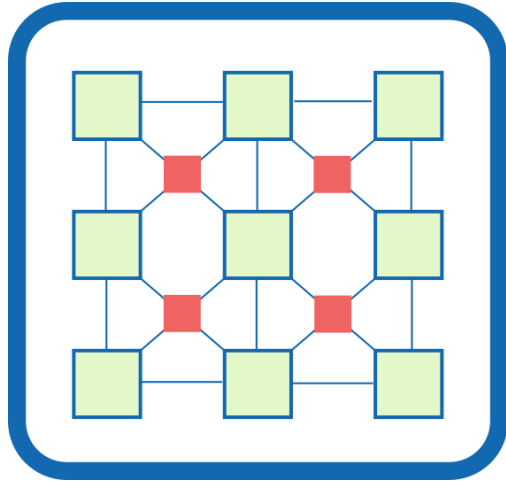
TIZIANO DE MATTEIS

# On the development of distributed applications with Intel FPGA SDK for OpenCL

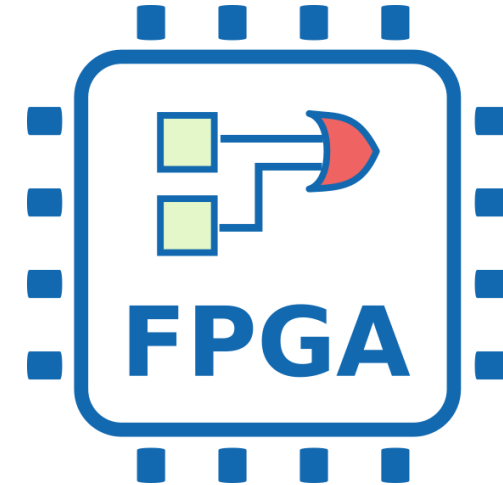
JOHANNES DE FINE LICHT, , JAKUB BERÁNEK, TAL BEN-NUN, TORSTEN HOEFLER AND OTHER AT SPCL, ETH







**Reconfigurable Hardware** is a viable option to overcome architectural von-Neumann bottleneck



Modern high performance **FPGAs** and **High-Level Synthesis** (HLS) tools are attractive for HPC



However, they are rarely considered in HPC: low productivity, lacks of re-usable components, very few real use cases

# Our contributions to the FPGA-HPC community

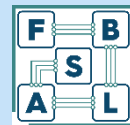
## Programming Model

To provide **performance portability** in  
Heter. Architectures  
(not only FPGA!)



## Libraries/Tools

To increase HPC  
programming  
**productivity**



## Applications

To show the potentials of  
reconfigurable hardware  
in **real applications**

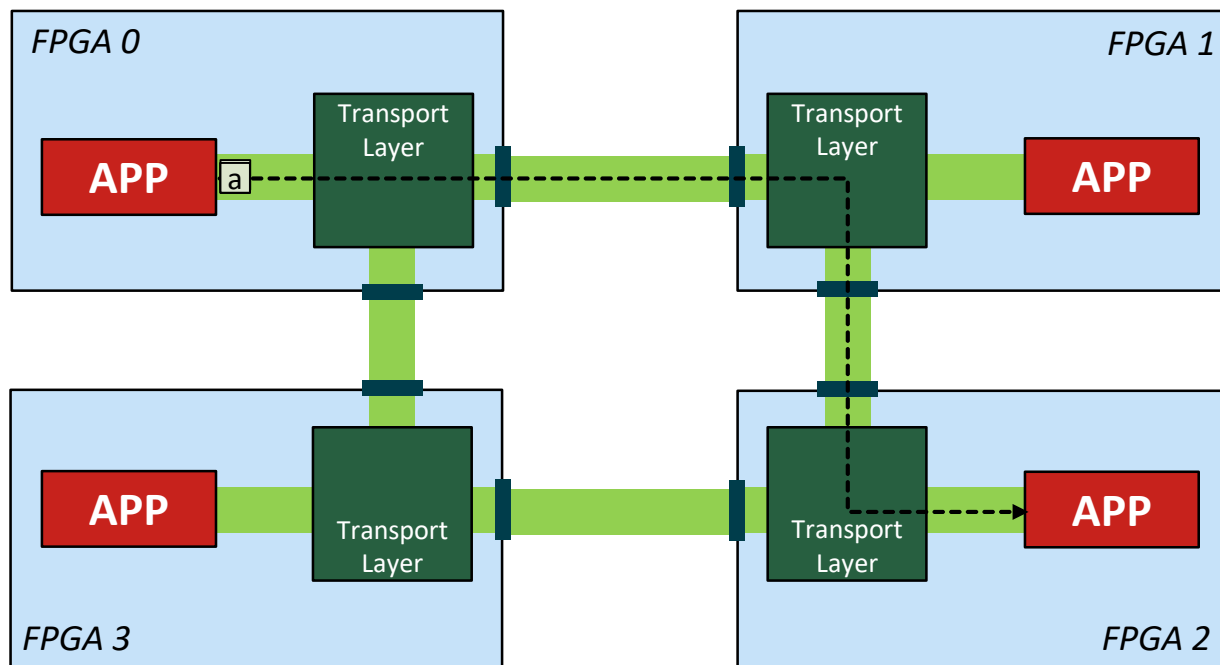
**StencilFlow**



# Streaming Messages – A Distributed Memory Programming Model for FPGAs

Traditional, buffered messages (MPI) are replaced with pipeline-friendly *transient channels*

```
Channel channel(N, my_rank + 2, 0); // Dynamic target
#pragma pipeline
for (int i = 0; i < N; i++)
    channel.Push(compute(data[i]));
```



Combines the best of hardware programming and message passing:

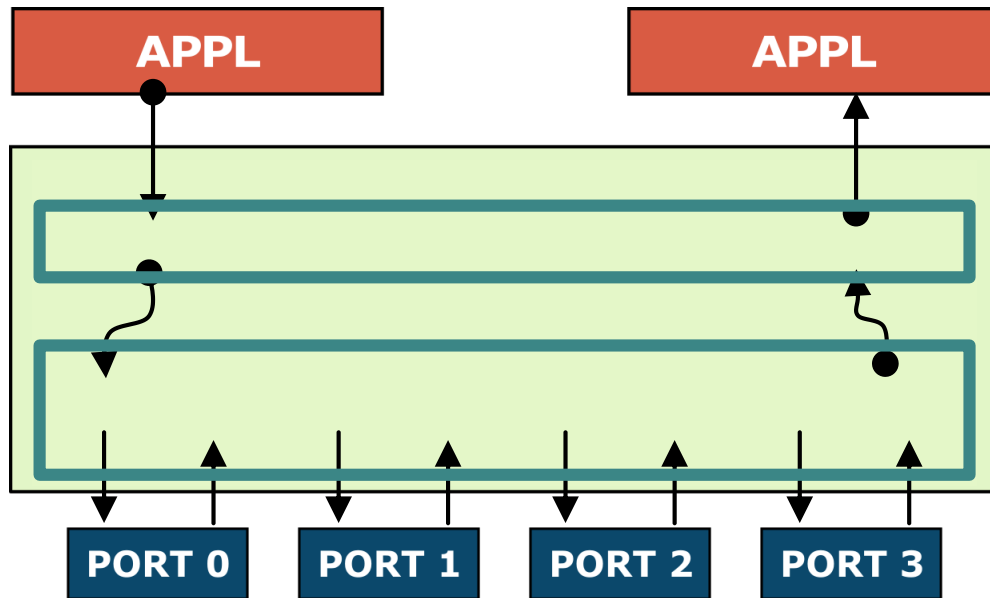
- Channels are transiently established, as ranks are specified dynamically
- Data is pushed to the channel during processing in a **pipelined** fashion

Key facts:

- Each channel is identified by a **port**, used to implement an hardware streaming interface
- All channels can operate in parallel
- Ranks can be programmed **either in a SPMD or MPMD fashion**
- No need to rebuild bitstreams if the topology (or number of ranks) changes

# Reference Implementation

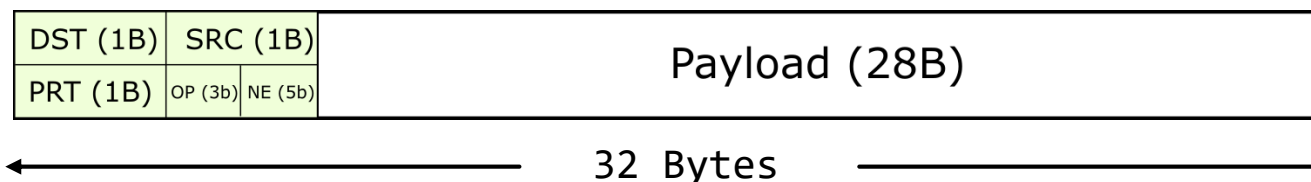
We implemented a proof-of-concept HLS-based implementation, targeting Intel FPGA ([github.com/spcl/smi](https://github.com/spcl/smi))



SMI implementation organized in **two main components**

**Port** numbers declared in **Open\_channel** primitives are used to lay down the hardware

Messages packaged in **network packets**, forwarded using packet switching on dedicated intra-FPGA connections

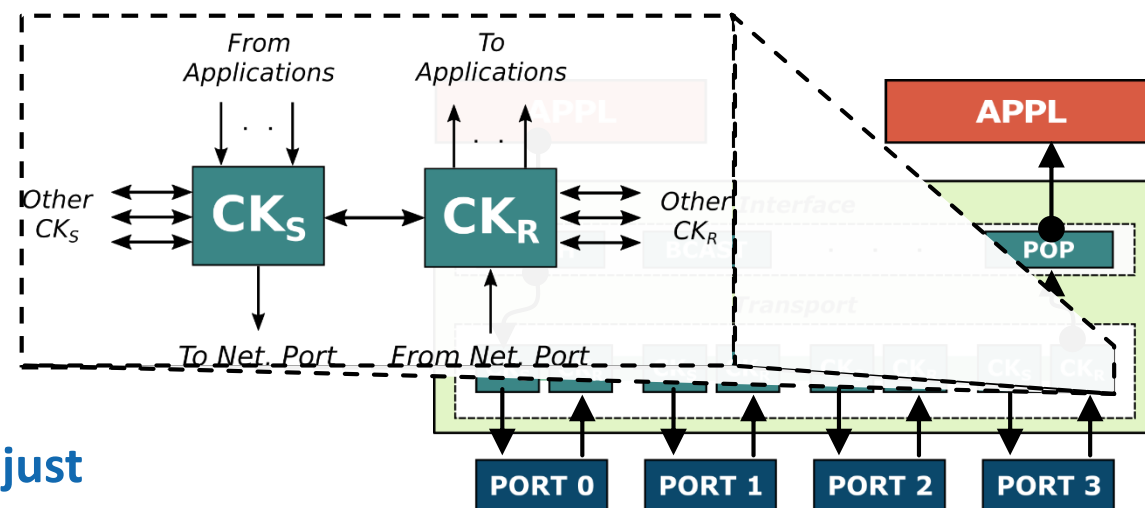


# Reference implementation – Intel FPGA SDK for OpenCL

Each FPGA net. connection is managed by a pair of  
**Communication Kernels (CK)**

Each CK has a dynamically loaded routing table that  
 is used to forward data accordingly

If the network topology or number of rank change, we just  
 need to rebuild the routing tables, not the entire bitstream

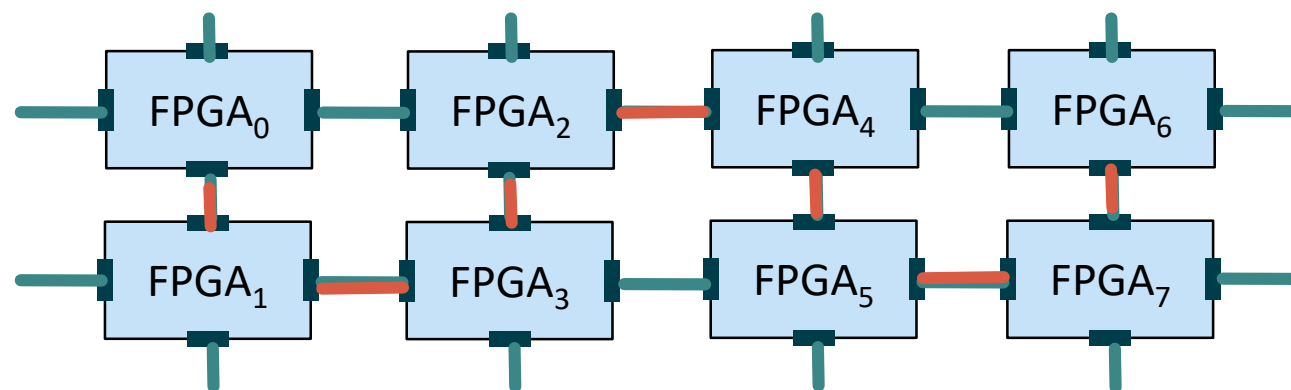


We had to deal with two types of *channels* (FIFO buffers)

- **On-chip channels:** the ones from/to applications and between Comm. Kernels  
**channel** long chan;
- **I/O channels:** network communications mapped to physical interface  
**channel** SMI\_Network\_message \_\_attribute\_\_((io("kernel\_output\_ch0")));

# Implementation

**Testbed:** Nallatech 520N boards (Stratix 10), each with 4x 40Gbit/s QSFP, offered as 8 I/O channels



Possibility to use different topologies and different number of ranks, by reconfiguring optical connections and by changing SMI routing

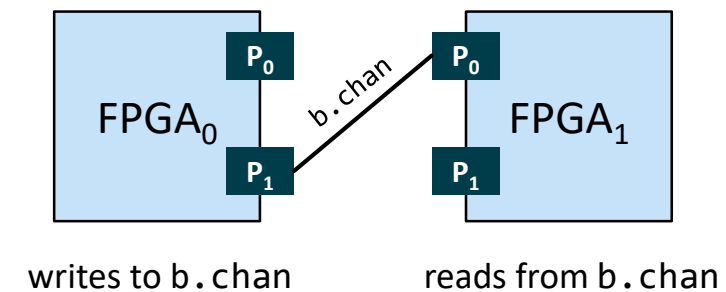
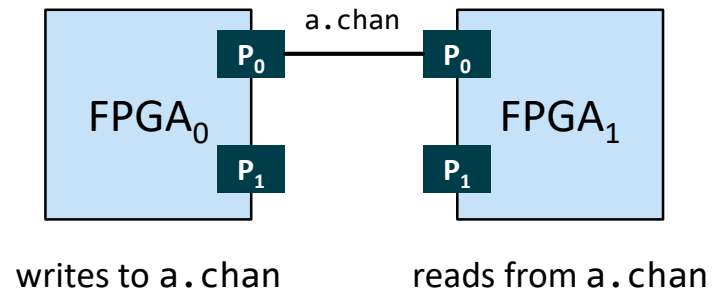
We needed a way to efficiently debug/emulating/running this, using different combinations of topologies, number of ranks, ...

# Emulation

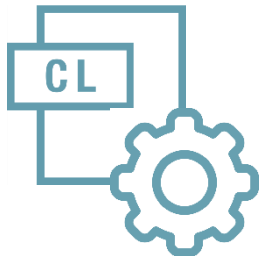
The Intel FPGA SDK for OpenCL offers a convenient way of emulating I/O channels, by means of files:

- an I/O input channel is emulated by reading from a file,
- and, and I/O output channel is emulated by writing to a file.

But we need to write a lot of code to evaluate even a single configuration:





Our approach:





# Development Workflow

-  User provided
-  SMI provided

```
#include<smi.h>
SMI_Push(
  &chan,
  &data
);
```

Source code

1. The **Code Generator** parses the user devices code and creates the SMI communication logic
2. The generated and user codes are synthesized. [For SPMD program, only one instance of the bitstream is generated](#)
3. A **Routes Generator** creates the routing tables (user can change the routes w/o recompiling the bitstream)
4. The user host program takes routing table and bitstream, and uses generated host header to start all SMI components

# Code-generation for Emulation – Device Code

```
#include <smi.h>

__kernel void App(int N, int root, SMI_Comm comm, /* ... */) {
    SMI_BChannel chan = SMI_Open_bcast_channel(N, SMI_FLOAT, 0, 1);
    int my_rank = SMI_Comm_rank(comm);

    #pragma pipeline // Pipelined loop
    for (int i = 0; i < N; i++) {
        int data;
        if (my_rank == root)
            data = /* create or load interesting data */;
        SMI_Bcast(&chan, &data);
        // ...do something useful with data...
    }
}
```

A:0 - B:0  
 A:1 - C:1  
 B:1 - C:2  
 ...  
**Topology**

\$ make app\_emulator

```
#include "smi/network_message.h"

// maximum number of ranks in the cluster
#define MAX_RANKS 8
//...

// QSFP channels
#if SMI_EMULATION_RANK == 0
channel SMI_Network_message io_out_0 __attribute__((io("emulated_channel_r0c0_r6c1")));
channel SMI_Network_message io_in_0 __attribute__((io("emulated_channel_r6c1_r0c0")));
channel SMI_Network_message io_out_1 __attribute__((io("emulated_channel_r0c1_r2c0")));
channel SMI_Network_message io_in_1 __attribute__((io("emulated_channel_r2c0_r0c1")));
channel SMI_Network_message io_out_2 __attribute__((io("emulated_channel_r0c2_r1c3")));
channel SMI_Network_message io_in_2 __attribute__((io("emulated_channel_r1c3_r0c2")));
channel SMI_Network_message io_out_3 __attribute__((io("emulated_channel_r0c3_r1c2")));
channel SMI_Network_message io_in_3 __attribute__((io("emulated_channel_r1c2_r0c3")));
#endif
#if SMI_EMULATION_RANK == 1
channel SMI_Network_message io_out __attribute__((io("emulated_channel_r1c0_r7c1")));
channel SMI_Network_message io_in_0 __attribute__((io("emulated_channel_r7c1_r1c0")));
...
#endif
```

Codegenerated

# Code-generation for Emulation – Host Code

```
#include <hlslib/intel/OpenCL.h>
#include "smi_generated_host.c"
//...

int main(...){
    MPI_Init(...);

    hlslib::ocl::Context context();
    auto program = context.MakeProgram(program_path);
    std::vector<hlslib::ocl::Buffer<char, hlslib::ocl::A
    SMI_Comm comm=SmiInit_broadcast(rank, rank_count, RO
        program, buffers);

    // create and launch app kernel ...
```

—————→ **\$ make app\_host**



```
SMI_Comm SmiInit_broadcast(int rank, int ranks_count,const char* routing_dir,...){

    const int ports = 1;
    const int cks_table_size = ranks_count;
    const int ckr_table_size = ports * 2;
    // load routing tables ...

    // create buffers for CKS/CKR and copy routing tables ...

    // start communication kernels...

    // return the communicator

}
```

**Codegenerated**

Then, what is left to do is to execute the application using the MPI Launcher:

```
$ env CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=8 mpirun -np 8 ./app_host emulator
```

(and keep your finger crossed )

# Stencilflow

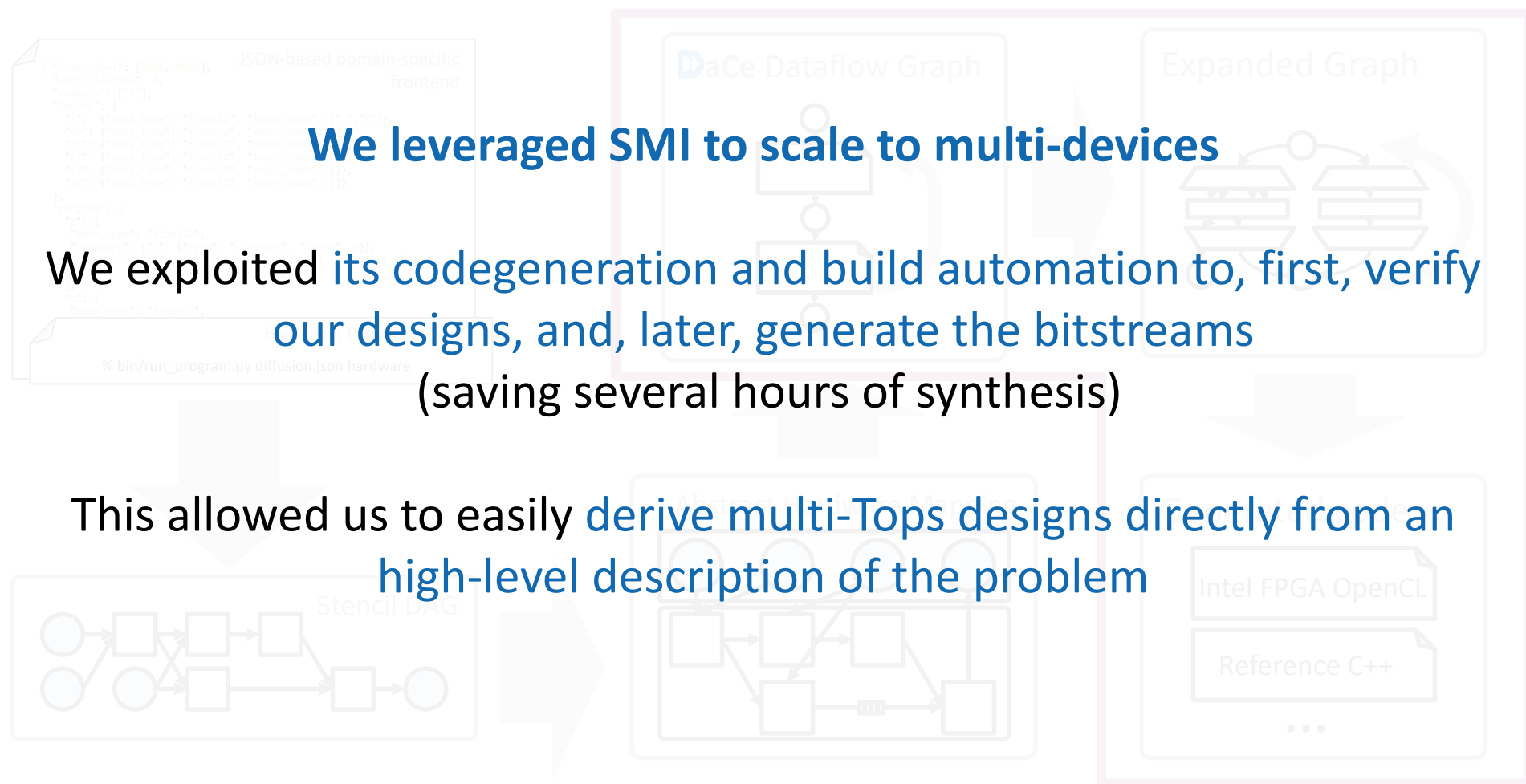
We apply the same solution also in Stencilflow, where all of this is further automated and abstracted-away





# Stencilflow

We apply the same solution also in Stencilflow, where all of this is further automated and abstracted-away



## Wrap-up

Code generation and build systems reduce developing time, and emulation is useful, but  
**it is not panacea**

If a (distributed) program emulates correctly, it does not mean that it will work in hardware:

- It may be not helpful to understand if there is a deadlock (because of its threaded execution model)
- The buffer depth is not the same of generated hardware
- It does not take into account the order of channel operations as implemented in the generated hardware
- It could take *forever*

## Wrap-up

### Emulation $\neq$ Debugging

Therefore we advocate for better development tools, that could ease programmers life:

- Precise emulation/simulation
- Full debugger
- The ability to inspect the status of (I/O) FIFO buffers
- Better reporting on the generated hardware

**We believe that these are necessary tools for HPC codes to start productively targeting reconfigurable (distributed) platforms**

# Thank You



[github.com/spcl/smi](https://github.com/spcl/smi)  
[github.com/spcl/dace](https://github.com/spcl/dace)  
[github.com/spcl/stencilflow](https://github.com/spcl/stencilflow)  
[github.com/definelight/hlslib](https://github.com/definelight/hlslib)

*spcl.inf.ethz.ch*  
*@spcl\_eth*

**ETH** zürich