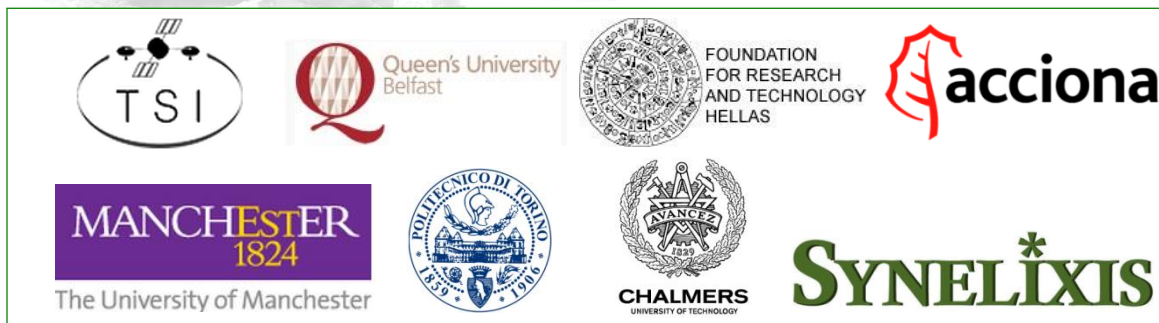


## Oil reservoir simulation in HPC

*Pavlos Malakonakis, Konstantinos Georgopoulos, Aggelos Ioannou, Luciano Lavagno, Ioannis Papaefstathiou and Iakovos Mavroidis*

**PRACEdays18**



# The Challenge

- Contemporary HPC trends shift towards exascale performance figures
  - ❖ Needed for modern-day compute-intensive and power-hungry applications like oil-reservoir simulation
    - ❖ 2 Highest performance HPC systems installed in industrial sites is in Total (Pangea - no 19 in Top 500) and Petroleum Geo-Services (Abel - no 24 in Top500)
      - ❖ Facebook is no 31 ☺ ☺
- Scaling in number of processing units alone does not suffice
  - ECOSCALE proposes a scalable programming environment *combined* with a novel hardware architecture to reduce *energy consumption* and *execution time*



# ECOSCALE Approach

- Processing takes place on *reconfigurable hardware*
- ECOSCALE-born architecture, namely *UNILOGIC*
- UNILOGIC offers sharing of all reconfigurable resources in a multi-FPGA comprised system
  - Significantly boosts the levels of parallelism in any given application
  - Combines the ability to move data close to the point of computation
  - Creates a novel platform for rapid algorithmic/task executions



# ECOSCALE Accelerator Cores

- Functionality in the reconfigurable hardware is implemented using *accelerator cores*
- They can be re-programmed to implement different functionality at any given time
- The cores are generated using a High-Level Synthesis (HLS) tool
  - OpenCL descriptions to generate RTL-level equivalent HDL IPs
- Test Case – Reservoir Simulation





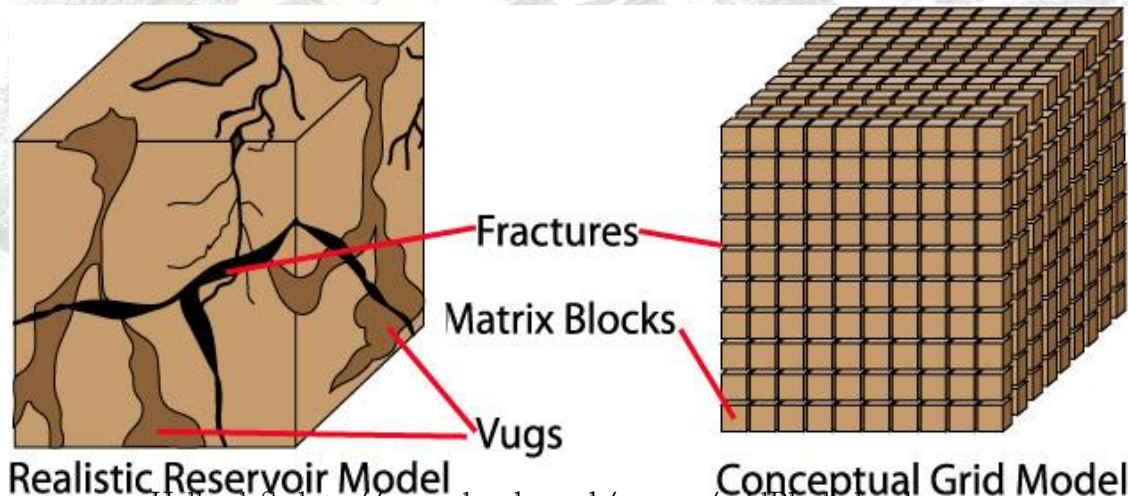
# Reservoir Simulation - Importance & Challenges

- Maximise the hydrocarbon recovery of an oil field
- Reservoir simulation is used to predict field performance under several possible production schemes
- Combines the physical laws applying to the oil production process, i.e.
  - ▶ mass and energy conservation
  - ▶ flow in porous media
  - ▶ thermodynamic phase equilibrium
- Differential equation numerical solution techniques
- Prediction of the future oil and gas production



# Reservoir Simulation- Overview

- Identifying the optimal field development scenario requires several simulation runs
  - Amounts to significant simulation time
- Computational tasks of such intensity can only be executed on workstations and dedicated clusters
- Reservoir rock matrix and its conceptual computer grid structure:



Holland, S., <http://www.dcs.gla.ac.uk/~samm/gridBlocks.html>



# Reservoir Simulation - Deeper

- Newton-Raphson (NR) method can solve the problem of modeling the flow of liquids deep underground
- Two algorithms utilized in the NR implemented
  - Hyperbolic
  - Michelsen
  - They both solve the Rachford-Rice equation
- Utilized in a Design-Space Exploration (DSE) process (both *manual* and *automatic*) for the FPGA-based ECOSALE HPC system



# The ECOSCALE System (I)

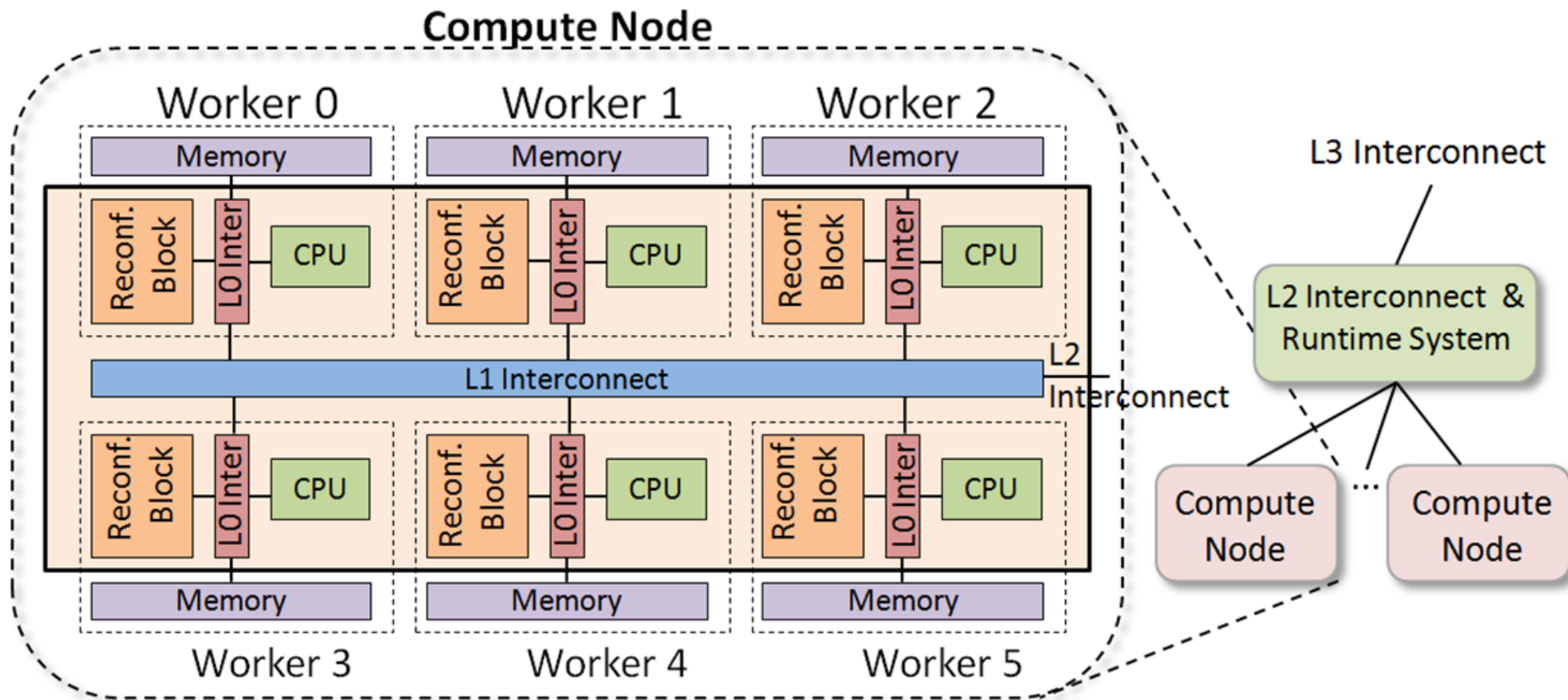
- Fundamental building block is the *Worker*
  - Many Workers exist within a single *Compute Node*
  - Each Worker is an independent processing node
  - Workers also feature:
    - a CPU that runs software routines that execute, fork and join tasks of an application
    - on-board DRAM
- The complete system comprises of multiple Compute Nodes





# The ECOSCALE System (II)

- Communication and synchronization between Workers is facilitated by a multi-layer interconnect



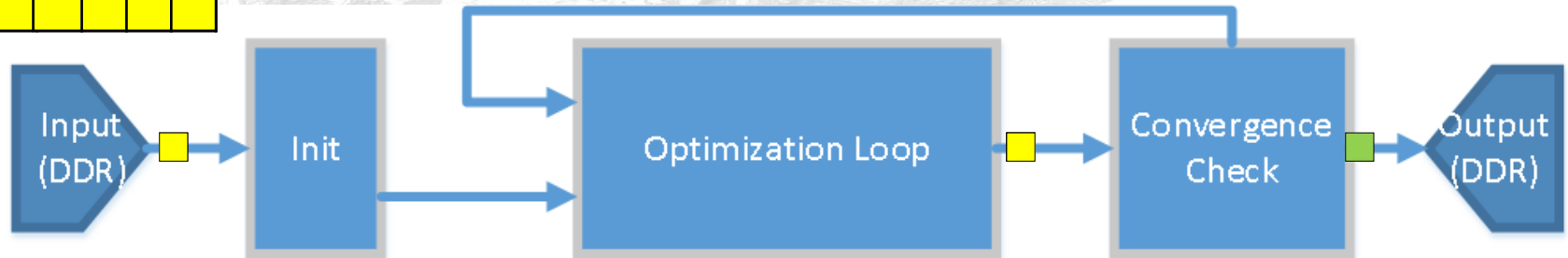
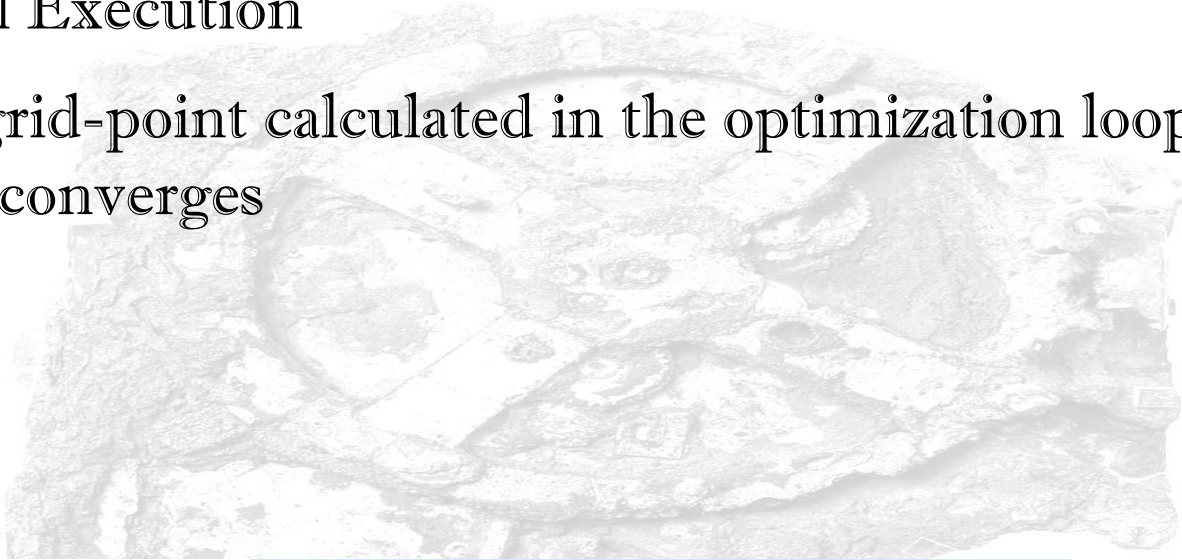
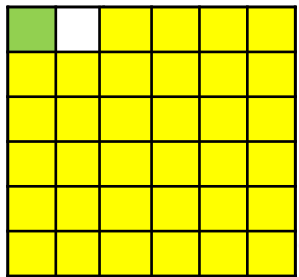
# How we program this monster ?

- Michelsen and Hyperbolic utilize a 3-dimensional grid-point setup
- The original OpenCL code performs grid-point computations sequentially, i.e. no parallelism
- Transformed code separates individual *optimization* problems that run in-parallel
  - *optimization* refers to reaching a desired level of accuracy for the coefficient under calculation
- In the original code, this is performed by sequential iterations inside a *while* loop



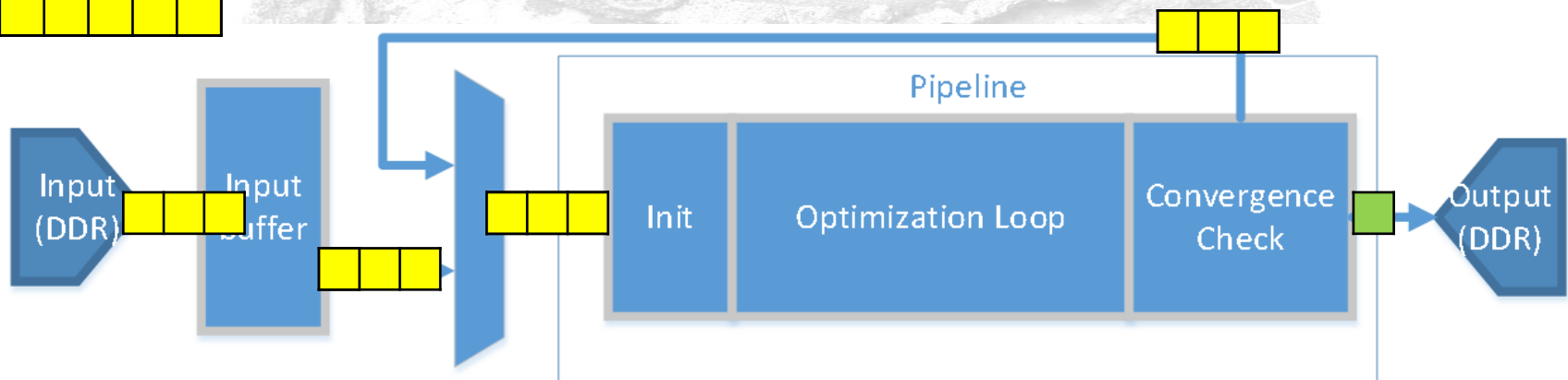
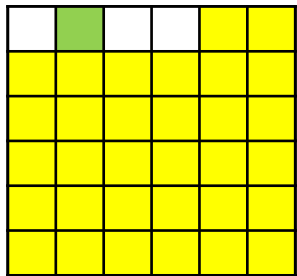
# Original Flow

- Original flow
- Sequential Execution
  - Each grid-point calculated in the optimization loop until the system converges



# Manual Optimization

- Optimized flow
- Modified it into a pipelined/parallel implementation by:
  - Modifying the *while* condition to run until all problems are solved instead of one reaching the desired accuracy
  - Executing this *while* condition for N grid points in-parallel where N is the Pipeline depth





# Manual Optimization - Code

```
int a
int b
for (all grid points)
    /*initialization*/
    a=amin
    b=bmin
    operation 3
    ....
    while(optimization
target)
        b=b+1
        a=a+1
        operation 3
        operation 4
        ....
    end while
    operation 1
    .....
    write result
end for
```



```
Buffer data
/*All variables become arrays*/
int a[pipeline_size]
int b[pipeline_size]
while (all problems solved)
    for (n : all grid points)
        if (initialization)
            /*initialization*/
            a[n]=amin
            b[n]=bmin
            operation 3
            ....
        end if
        else /*while loop code*/
            b[n]=b[n]+1
            a[n]=a[n]+1
            operation 3
            operation 4
            ....
            If (optimization reached)
                operation 1
                .....
                write result[n]
                solved++
            end if
        end else
    end for
end while
```

- Pseudocode is the same for both algorithms



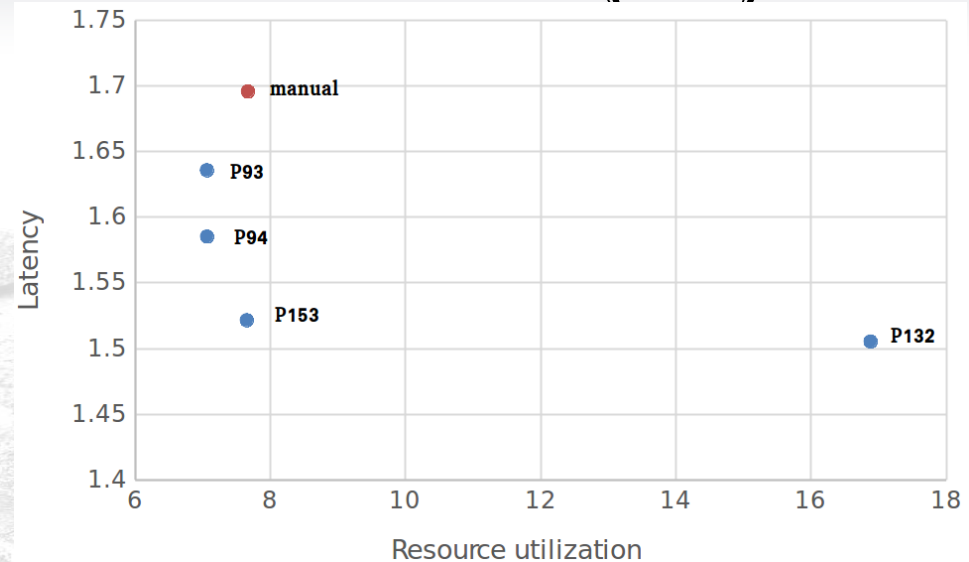
# Automated Optimizations (I)

- Implemented on top of the *Xilinx Vivado HLS* tool
- Modified code has been annotated to help the tool generate optimized implementations
- Subsequently, the DSE process has performed a number of (transparent to the user) steps, i.e.
  - Identify code areas where directives can improve performance
  - Generate a set of DSE exploration options of different directive combinations
  - Execute set of parallel runs so as to evaluate the combinations
  - Decide on the best candidates



# Automated Optimizations (II)

- Pareto-optimal solution for the Hyperbolic algorithm
- DSE process explores 256 points in the design space
- Optimal points

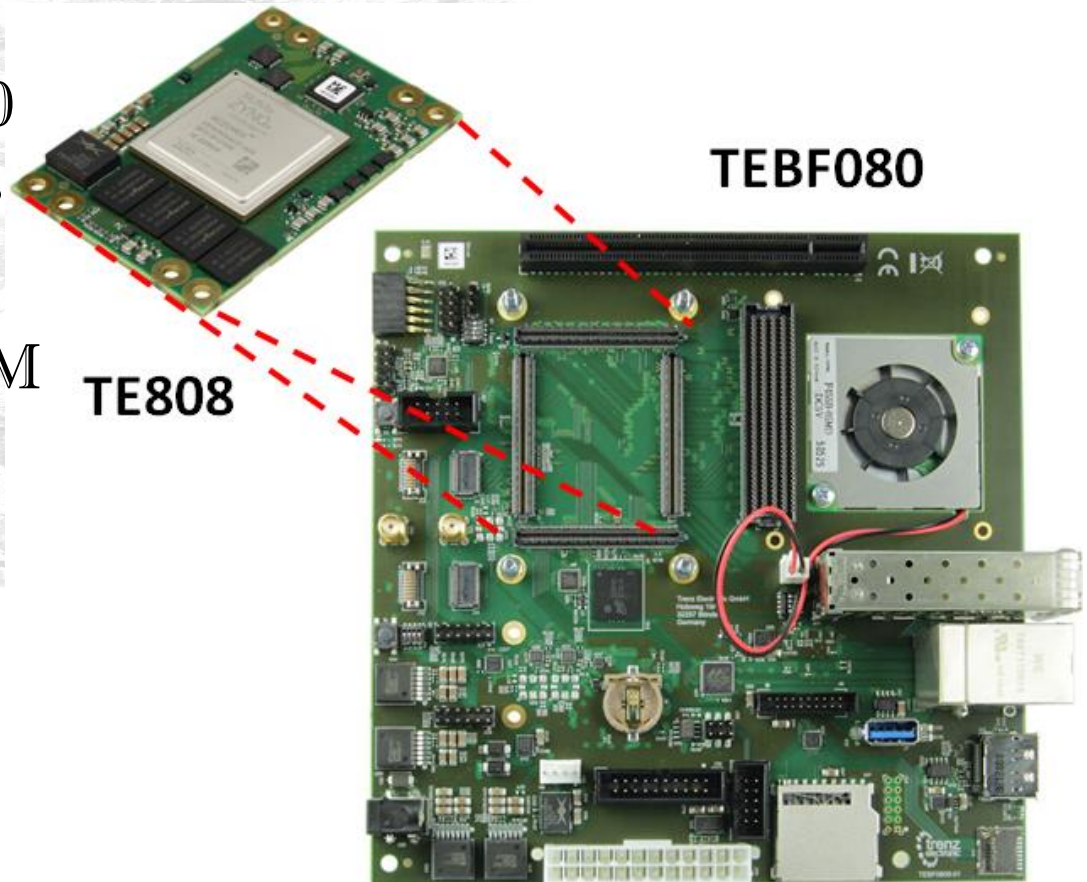


- Red dot is the best performance achieved manually
- Points 94 and 93 strong candidates since they offer close latency but at reduced resource utilization
- Lower latency does not necessarily mean worse performance since it may lead to increased throughput
- Eventually, implemented solution has come from merging the manual and P153 points



# Evaluation (I)

- IP accelerator cores have been tested on the initial ECOSCALE prototype
- Single Worker
  - TE808 and TEBF080  
Trenz Electronic boards
  - UltraScale+ MPSoC  
FPGA with 2GB DRAM
  - Quad-core ARM  
Cortex-A53 platform
  - Dual-core Cortex-R5  
real-time processor





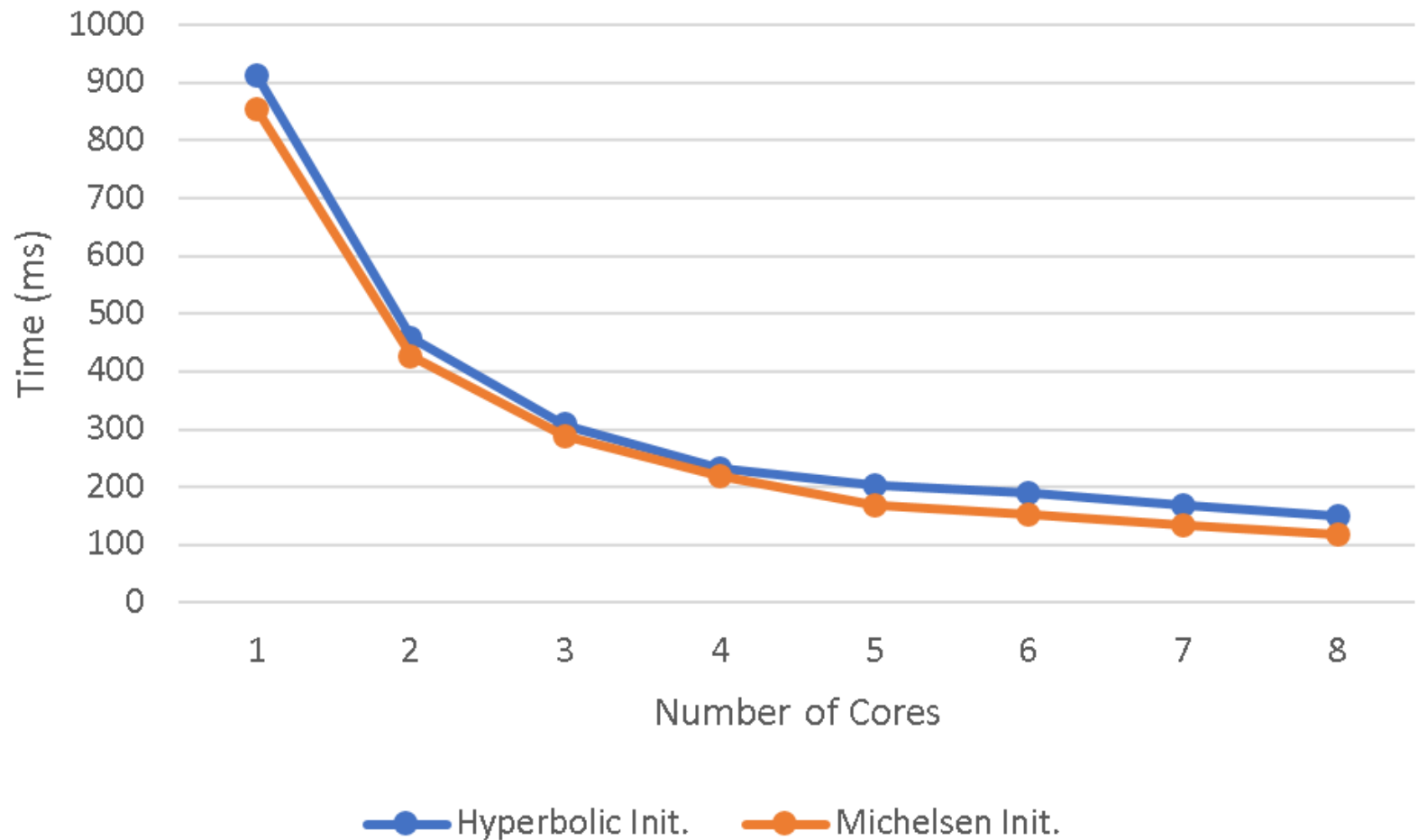
# Evaluation (II)

- Two different data sets used, i.e. 100K and 200K grid points
- Measured times compared against those from an i5 Quad-core CPU at 3.1 GHz
- Results shown for different numbers of in-parallel hardware cores
- Results for *i)* the *initial* (non-optimized raw OpenCL code) HLS-generated cores and *ii)* the manual and automatically *optimized* HLS-generated cores



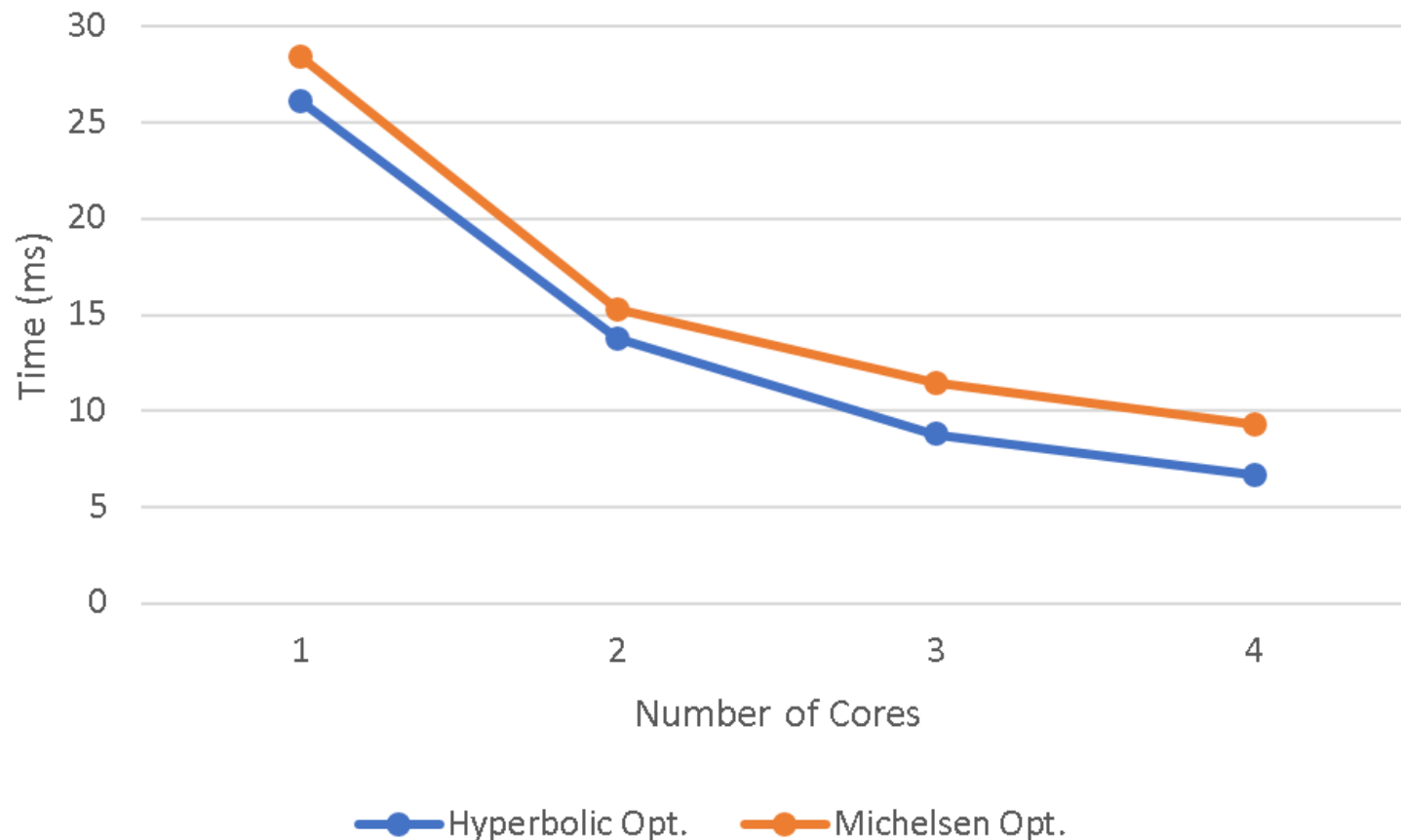
# Evaluation (III)

- Results for the *initial* accelerator IPs with varying number of cores



# Evaluation (IV)

- Results for *optimized* cores (a maximum of four can fit in the reconfigurable fabric)



# Evaluation (V)

- Optimized cores execute considerably faster
- Execution time drops with the increase of the number of in-parallel cores until data transfer time becomes the bottleneck
- Main FPGA can accommodate a maximum of four optimized accelerator cores
- Performance has been compared against conventional processing methods
  - CPU with either single or four-thread execution using OpenMP





# Evaluation (VI)

	Software			
Platform	CPU		CPU OpenMP (4 thread)	
Data size	100K	200K	100K	200K
Hyperbolic	25.5	50	8.8	17.2
Michelsen	29	56	11.7	22.3

	Hardware			
Platform	Initial 8 Core		Optimised 4 Core	
Data size	100K	200K	100K	200K
Hyperbolic	151	298	6.7	13.8
Michelsen	117	235	9.3	18

	speedup opt. vs. init	speedup opt. vs. 4-thread	speedup opt. vs. 1-thread	efficiency vs. 4-thread
Hyperbolic Opt	22.5	1.3	3.8	13
Michelsen Opt	12.5	1.2	3.1	12



# Conclusions

- Execution on reconfigurable hardware yields better performance than CPU
  - Optimized cores offer a speedup of 3,8 and 3.1 over the single-threaded CPU execution for Hyperbolic and Michelsen respectively
  - ECOSCALE triggers a 20%-30% speedup over a 4-thread execution
- ECOSCALE yields an order of magnitude better energy efficiency
  - CPU comes at a typical 77 Watts power consumption
  - Average US+ FPGA power consumption is 7.8 Watts



FPGA-based HPC is indeed very  
promising and quite easy to  
explore!

