www.fpgafais.com

Supercomputing with FPGAs

Dr Grzegorz Korcyl

Department of Information Technologies

Jagiellonian University, Cracow

Dr Piotr Korcyl

M. Smoluchowski Institute of Physics

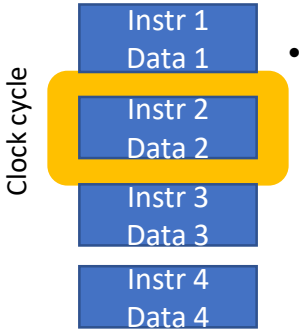
Faculty of Physics, Astronomy and Applied Computer Science

Jagiellonian University, Cracow

Supercomputing Frontiers Europe 2019, Warsaw, 11-13 March 2019

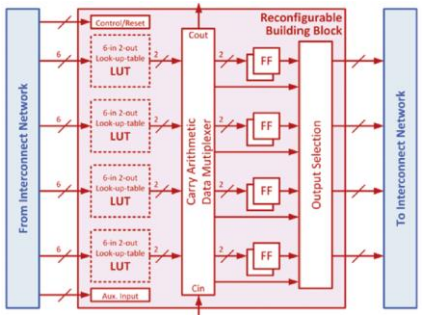
Why FPGAs?

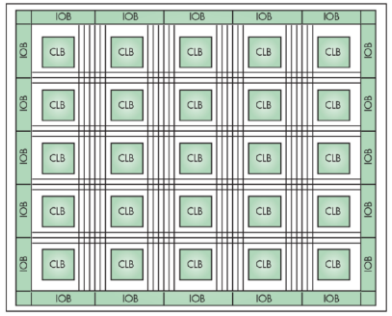
- Field Programmable Gate Arrays
  - Arrays of Configurable Blocks
  - Reconfigurable at any time
  - Intensive growth: 4x over 4 years
  - Available resources <-> performance
  - High-level development methodologies
  - Wide range of available platforms

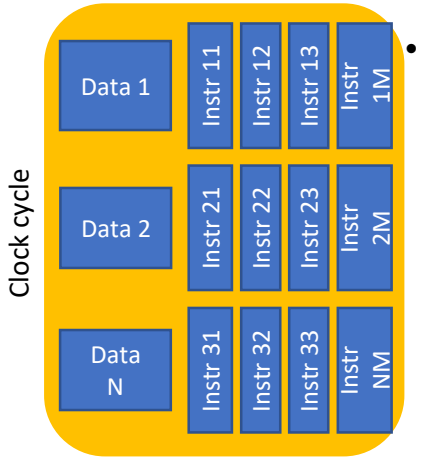


- CPU
  - Use given architecture
  - Single Instruction Single Data per Core
  - Multiple Cores
  - High clocking frequency
  - Operating system

- Development of algorithms faster than hardware
  - Configurable / adaptive hardware
- Between general purpose units and application specific circuits





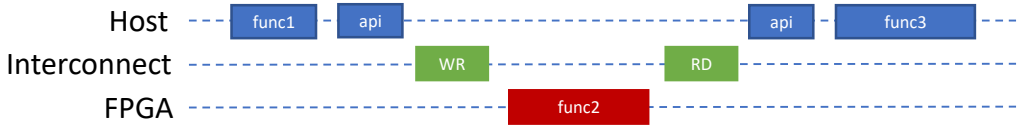


- FPGA
  - Design optimal architecture for a given problem
  - Natural parallelism
  - Streamlined processing
  - Instant memory access
  - Standalone platforms
  - Low clocking frequency

www.electronicdesign.com

Acceleration on FPGAs

- Delegate computational intensive functions from HOSTs to KERNELs in Programmable Logic
  - Profit from:
    - Pipelined processing
    - Natural parallelism
    - Distributed internal memory, direct DDR connection
    - Embedded network transceivers
    - Deterministic latency
    - Low power consumption
  - Bottleneck - data delivery



- Cloud providers (FPGA as service):
  - MS Azure - 2017
  - Amazon AWS – Sep. 2017
  - Huawei – June 2018
  - Aliyun – Sep. 2018
  - Nimbix – Oct. 2018

- Cyfronet Kraków - 2013

- Architectures:
  - Classic FPGA
    - Host: x86 CPU
    - Interconnect: from host DDR memory through PCIe to DDR on FPGA platform
  - System-on-Chip
    - Host: ARM cores embedded within FPGA package
    - Interconnect: DMA transfers with shared DDR memory

Kernel implementation

- Conjugate Gradient solver as a benchmark for HPC systems
  - Accelerate sparse matrix-vector multiplication
    - Low level of data dependency
    - High level of parallelization
    - Large problem sizes

1. Host initiates the algorithm, reads input data from storage
2. Prepares dataset for the kernel
3. Transfers data and calls the kernel
4. Retrieves results and evaluates next iteration

- Kernel development
  - High-Level Synthesis
    - Engine to generate RTL from C/C++, OpenCL sources
    - Wide range of #pragmas and OpenCL attributes to control RTL production
    - Frameworks to import Tensorflow/Keras to HLS
    - Detailed reports describing compilation results
- System development
  - Define data transfer infrastructure between Host and Kernel
  - Compile and generate executable files

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33	2.820	0.90

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2097	2097	2097	2097	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	1568	1568	290	1	1	1280	yes
- data_out_transfer	514	514	4	1	1	512	yes

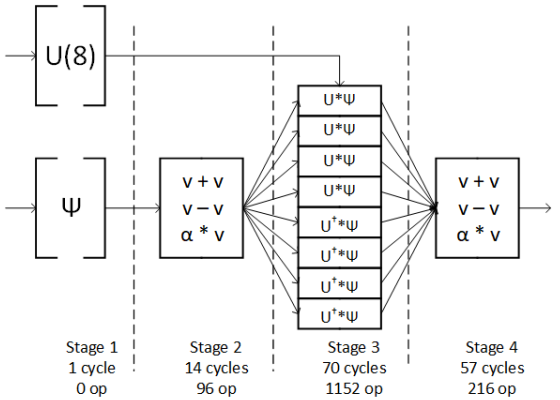
Utilization Estimates

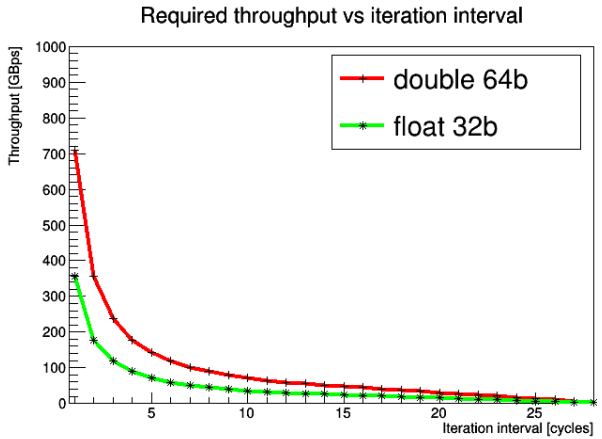
Summary

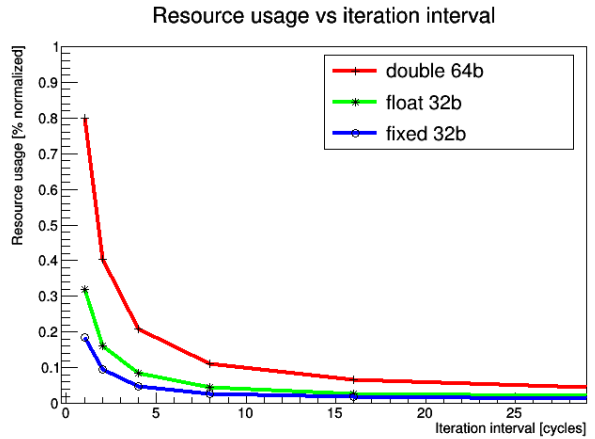
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	237	-
FIFO	-	-	-	-	-
Instance	150	2596	505116	290902	-
Memory	0	-	512	0	8
Multiplexer	-	-	-	352	-
Register	0	-	24049	320	-
Total	150	2596	529677	291811	8
Available	5376	12288	3456000	1728000	1280
Utilization (%)	2	21	15	16	~0

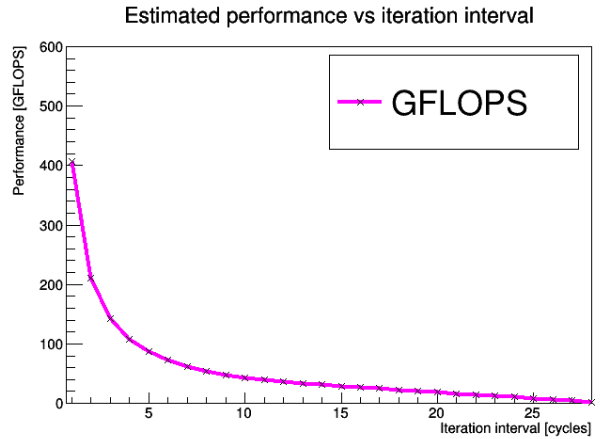
Kernel details

- High-level synthesis optimizations
  - Balance between: Latency / Iteration Interval / Resources
  - Ensure full pipeline
    - Accept new data set considering maximum throughput from memory
  - Loop unrolling for parallelization
  - Internal memory management
    - Ensure entire required data set is available at a given clock cycle
  - Data type selection
    - Custom bit-width data types
- Kernel summary:
  - 1464 FLOP per iteration
  - 296 variables required to start the iteration
  - Compiled for Xilinx U250 with 300MHz clock





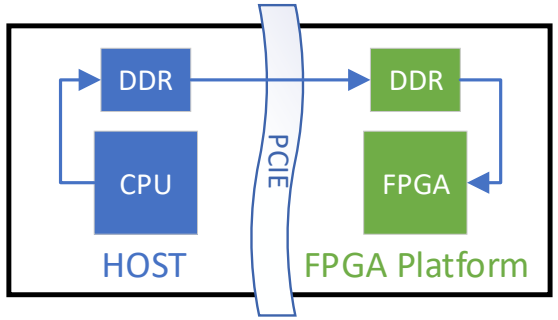


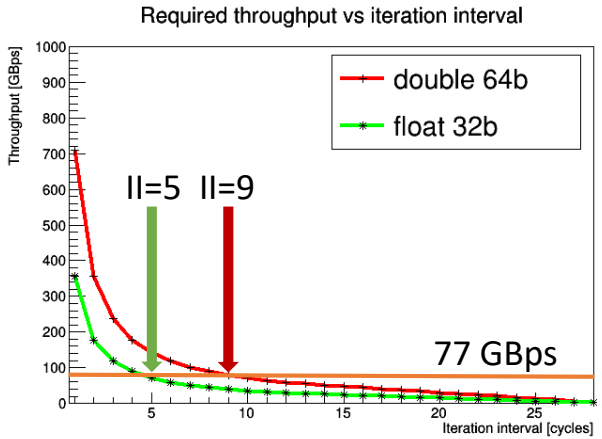


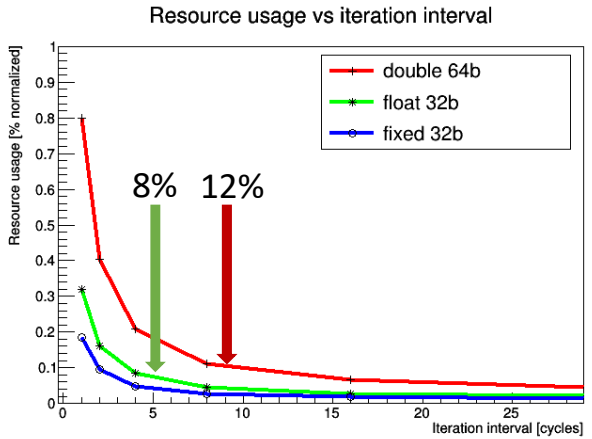
External memory

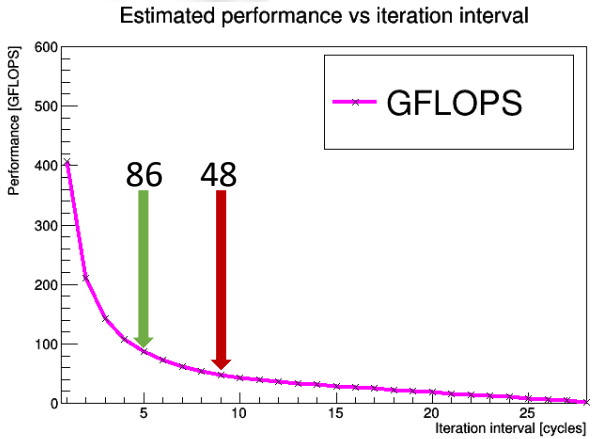
- Store data in DDR connected to the FPGA
  - Large capacity
    - Many data sets stored for pipelined iterations
  - Limited bandwidth and additional latency
- Implemented on Xilinx Alveo platform
  - Initial data preparation on x86 host
  - Data transfer over PCIe
  - DDR 64GB capacity – problem size 49152*8*8*8 (double)
  - 4x DDR 512b wide @ 300MHz = 77GBps
  - Single kernel instance exceeds memory bandwidth

limit







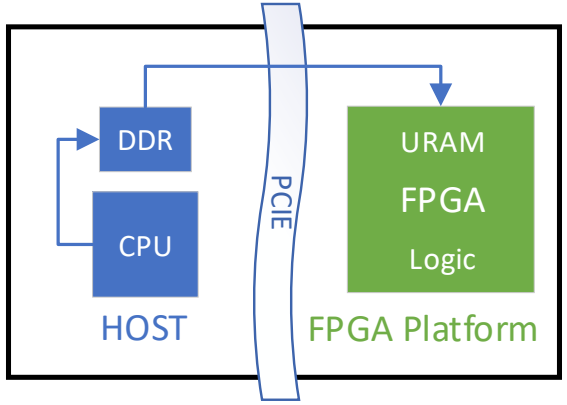


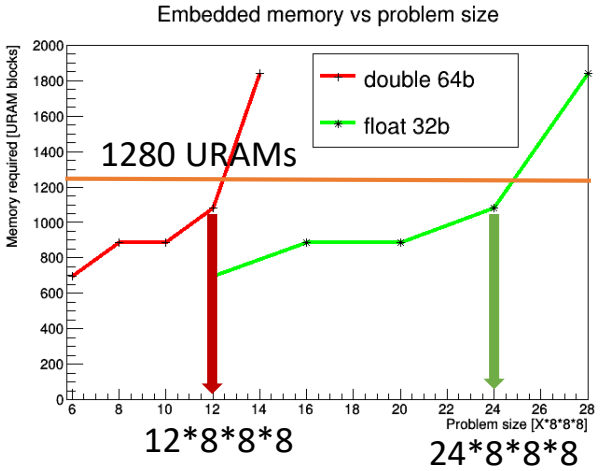
Embedded memory

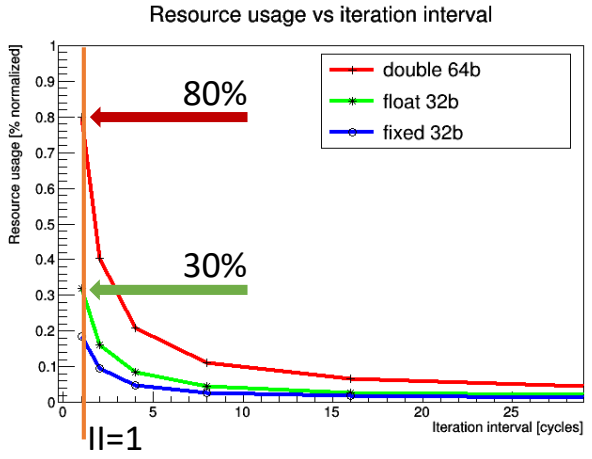
- Use embedded, distributed memory resources
  - Small capacity – frequent in/out data transfers
  - Very high bandwidth
  - Single-chip, standalone, low-power solution
- Simulated for Xilinx Alveo platform

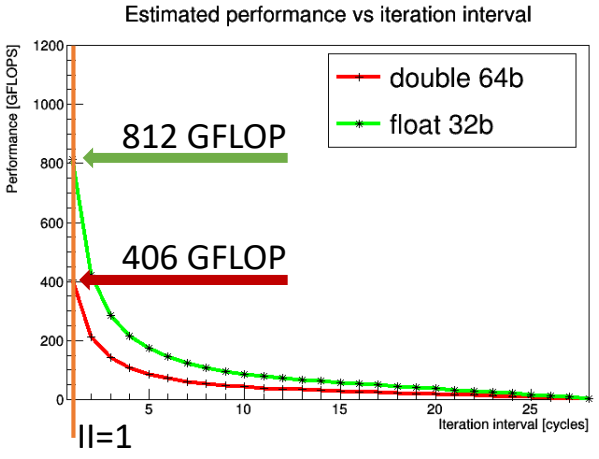
limit

  - 54 MB memory – problem size 12*8*8*8 (double)
  - 32 TBps bandwidth
  - Kernel with iteration interval 1 achieveable









Conclusions and future prospects

- **FPGAs are versatile platforms suitable for HPC applications**
  - Software only development (FPGA architecture understanding helpful)
  - Commercially available platforms
  - Reprogrammable within ms – self adapting platforms, power savings

  - Single-clock-cycle Conjugate Gradient kernel developed and measured with two data delivery methods:
    - External memory: 48 GFLOPs (double), 86 GFLOPs (float)
    - Embedded memory: 406 GFLOPs (double), 812 GFLOPs (float)
  - Current state-of-art CPU implementations:
    - Intel Broadwell 6 cores: 47 GFLOPs (double), 95 GFLOPs (float)
    - Intel KNL 64 cores: 220 GFLOPs (double), 345 GFLOPs (float)

S. Durr, *„Three Dirac operators on two architectures with one piece of code and no hassle”,* LATTICE2018, arXiv:1808.05506v2, Nov. 2018

- **Possible improvements**
  - External memory
    - FPGA with integrated HBM: 8GB capacity, bandwidth up to 460 GBps (Xilinx Alveo U280 available Q3 2019)
  - Embedded memory
    - Employ integrated high-speed serial transceivers to deliver data directly to the kernel instances
      - Up to 120x 32.75Gbps links (aggregated 480 GBps bandwidth on XCVU9P)

This work was in part supported by Deutsche Forschungsgemeinschaft under Grant No. SFB/TRR 55 and by the polish NCN grant No. UMO-2016/21/B/ ST2/01492, by the Foundation for Polish Science grant no. TEAM/2017-4/39 and by the Polish Ministry for Science and Higher Education grant no. 7150/E-338/M/2018.
The project could be realized thanks to the support from Xilinx University Project and their donations.