Study and Analysis of Hybrid CPU-FPGA computing

EE 669: R n D presentation

Name: Yatish Turakhia

Roll No: 09D07015

Von Neumann Paradigm

- Backbone of modern general purpose computing.
- Consists of the following:
  - Separate memory for storing data and instructions
  - A control unit (PC, Decoder etc)
  - An arithmetic and logic unit (datapath)
  - I/O to communicate with outside world
- Major advantage: flexibility. Can program almost any existing algorithms!

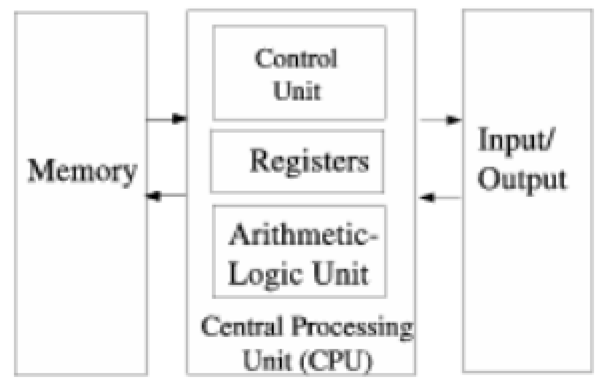


Figure 1. Von Neumann Architecture

Von Neumann Paradigm: Limitations

- Data moves between memory requires instruction streams.
- Execution of operations requires fetching and decoding of instructions.
- Instruction streams memory-cycle-hungry.
- Sequential execution: not most efficient!!
- Excessive power consumption!

Trends in modern day processors

- Transistor density continues to rise exponentially.
- Power density rising exponentially too [2][16]. Future processors to be power-limited. More transistors than can be simultaneously switched on.
- Performance scaling harder to achieve. Need to reinvent computing!!
- Increasing focus on heterogeneous computing: application-specific acceleration using dedicated hardware accelerators [16] or heterogeneous CMPs with different processors [1][11].
- Strong trend towards hybrid architectures that combine FPGAs and general purpose processors [6][7][8][15]

Reconfigurable computing: the hardware paradigm

- Field Programmable Gate Arrays (FPGAs) are programmable semiconductor devices that are based around a matrix of configurable logic blocks connected via programmable interconnects .

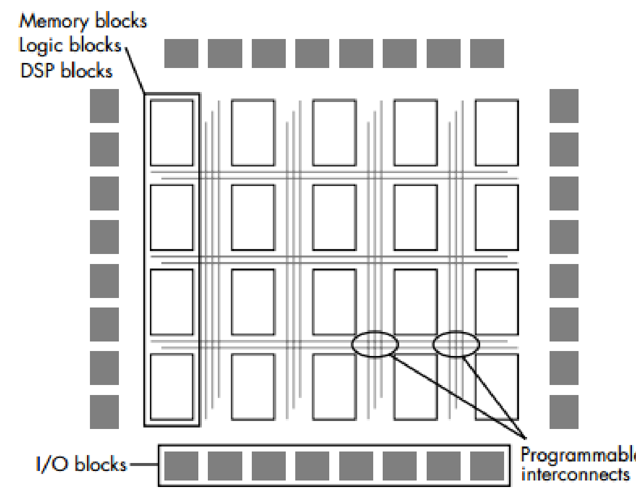


Figure 2. Typical FPGA architecture

FPGA Advantages

- Highly customizable. Custom pipelining.
- Highly parallelizable: Can replicate a hardware function block multiple times.
- Primarily driven by data streams: no instructions!
- Much more power efficient in comparison to CPU. Per area power consumed 10-100 times better for FPGA compared to a standard Intel Core[6].
- Scalable roadmap! [14]
- Orders of magnitude of speedup over standard processors [8], [23] with 10-40X lower clock rate. The "Reconfigurable Computing Paradox "[22]

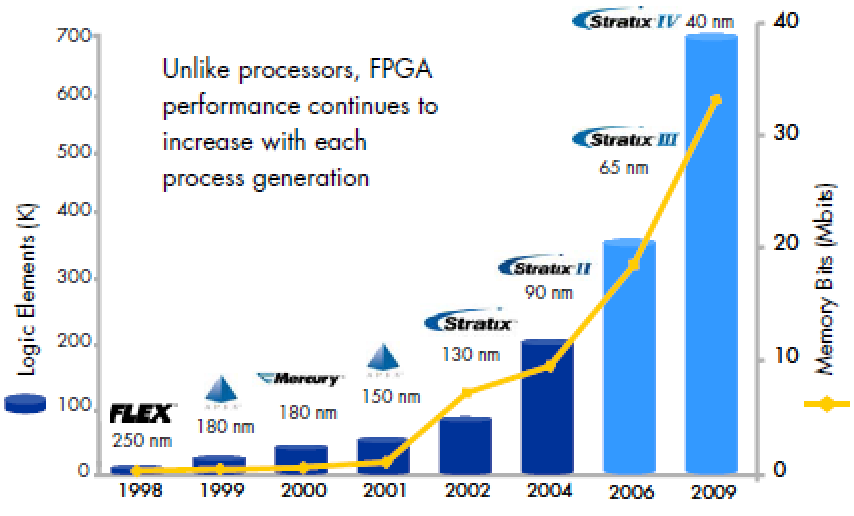


Figure 3. FPGA density. Source: [14]

Major Contributions

- Study and analysis on CPU-FPGA computing and tools.
- Study and implementation of algorithms using high level synthesis tools like C-to-Verilog[3].
- Study and implementation of algorithms using hardware-software co-design using SIRC Application Programming Interface (API).
- Developing a complete framework to easily implement algorithms on CPU-FPGA interface using SIRC.
- Preparing a roadmap and groundwork for further study in this direction.

CPU-FPGA Co-processing

- Divide the applications computations into a set of software instructions executing on the processor and a portion which runs on an FPGA [7]
- The timing-critical portions of the design should execute on the faster FPGA while non-critical portions should run on the processor [6][7].
- Communication necessary between processor and hardware (FPGA). Need for an interface between CPU and FPGA.

High Level Synthesis

- High programming cost associated with FPGA. Requires low level RTL specification for synthesis.
- HLS: Automated generation of RTL design from a high level behavioral specification while satisfying design constraints and optimizing on the given cost function [9].
- Several open source HLS tools available: LegUp [18], C-to-Verilog [3] etc. Commercial ones include Xpilot [19], Pico [20] etc

High Level Synthesis

- A typical HLS tools uses dataflow analysis to extract parallelism, perform loop unrolling, implement clocking strategy and data path and control path allocation.
- We used C-to-Verilog to implement following two algorithms on Xilinx Virtex XUPV5 board:
  - Addition of 2 input arrays.
  - Discrete Wavelet Transform

Simple Interface for Reconfigurable Computing (SIRC)

- Simple, open-source communication API between CPU and FPGA [12][13].
- A simple and easy to use high-level communication and synchronization protocol.
- Abstracts way implementation details. No need to worry about drivers, OS or communication protocol.
- Need only basic knowledge of C++ and Verilog for usage.
- Communication over Gigabit Ethernet. Ethernet is unreliable, but SIRC uses TCP-like ACK/retry mechanism to provide reliable communication

SIRC: Overview

- Uses two APIs: one on software side and one on hardware side. Master Slave arrangement. Host PC is master and FPGA slave.
- Software code:
  - Sends data from host to FPGA using an input buffer which is connected to FPGA
  - Signals the FPGA to begin its execution on the input data
  - Waits till the FPGA completes its computation on the input data
  - Receives the output data from the FPGA using an output buffer which is also connected to the FPGA

SIRC: Overview

- Hardware Execution:
  - Waits till it receives a signal from the host to begin execution.
  - Receives the input data from the input buffer connected to host and starts processing. On completion, it puts the result data on the output buffer.
  - Signals to host that it has completed its execution and goes back to idle state.

SIRC: Overview

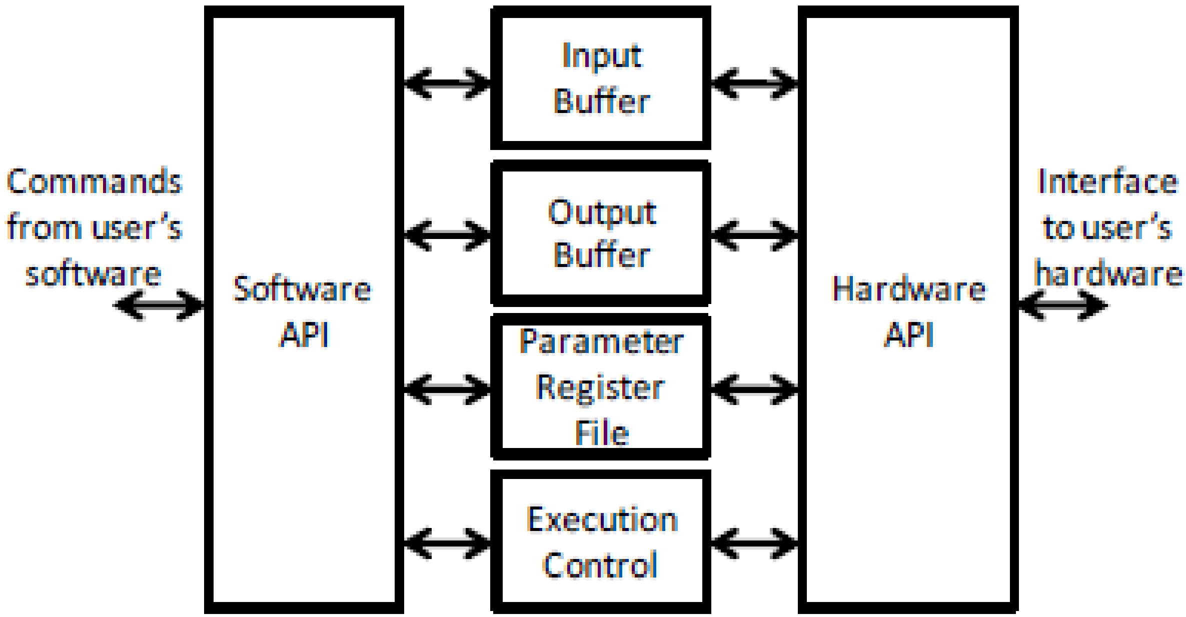


Figure 4. SIRC software and hardware API. Source [12]

- SIRC also allows creation of multiple API interfaces to overlapped I/O and hide latencies
- Provides bandwidth ~450 Mbps for small transfers and upto 950 Mbps (98% theoretical maximum) for transfers above 512KB.

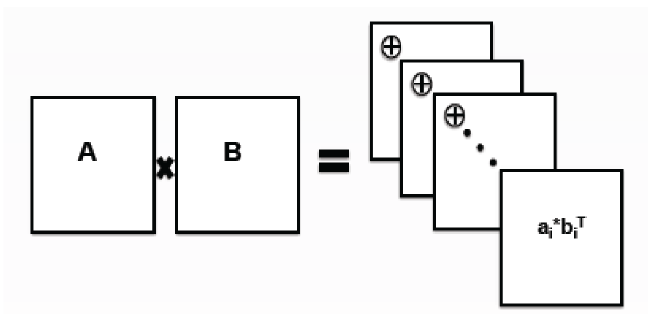
Matrix Multiplication GF(2)

- GF(2): Finite field containing 2 elements. Smallest possible finite field.
- Applications in coding theory, cryptographic algorithms, matrix inversion [4] etc.
- GF algorithms better suited on FPGA. No instructions on GF operations in CPU.

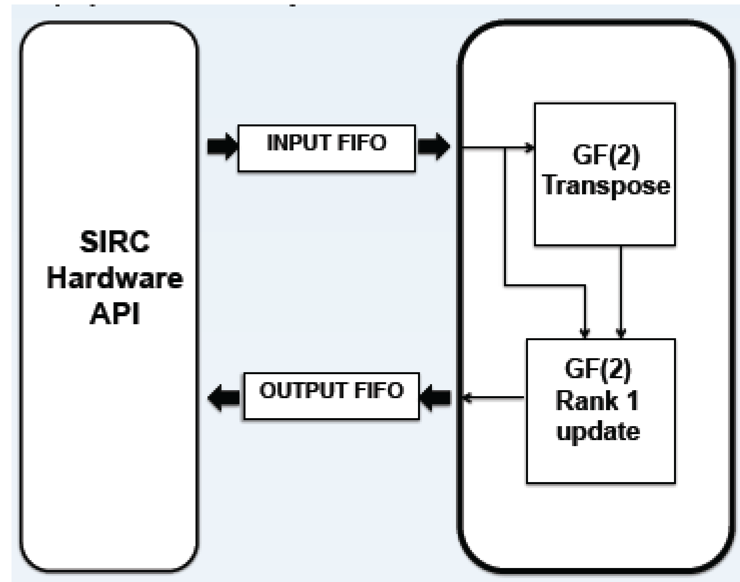
Matrix Multiplication GF(2): Implementation Details

- Used Verilog based synthesizable code for performing multiplication of two 32 bit x 32 bit matrix blocks. Multiplier has 2 components: transpose and rank 1 update.
- Multiplication of larger matrices by block multiplication algorithm [5].
- Block reuse to minimize communication. Send one block of matrix A and row of blocks for matrix B.
- Output blocks sent to PC via output buffers where accumulate operation takes place.
- Used input and output FIFOs to interact with hardware API of SIRC. Simplifies hardware programming significantly.
- Software C++ code running on 64-bit 2.66 GHz Intel i7 processor. Hardware on Xilinx Virtex XUPV5 FPGA.

Matrix Multiplication GF(2): Implementation Details



(a)



(b)

Figure 5. (a) Multiplication of 2 matrix blocks (b) GF(2) multiplication using SIRC H/W API on FPGA

Matrix Multiplication GF(2): Implementation Details

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	3,581	69,120	5%	
Number used as Flip Flops	3,581			
Number of Slice LUTs	8,327	69,120	12%	
Number used as logic	5,489	69,120	7%	
Number using O6 output only	5,029			
Number using O5 output only	160			
Number using O5 and O6	300			
Number used as Memory	2,827	17,920	15%	
Number used as Dual Port RAM	2,816			
Number using O6 output only	2,816			
Number used as Shift Register	11			
Number using O6 output only	11			
Number used as exclusive route-thru	11			

Matrix Multiplication GF(2): Implementation Details

Number of occupied Slices	2,807	17,280	16%	
Number of LUT Flip Flop pairs used	8,739			
Number with an unused Flip Flop	5,158	8,739	59%	
Number with an unused LUT	412	8,739	4%	
Number of fully used LUT-FF pairs	3,169	8,739	36%	
Number of unique control sets	165			
Number of slice register sites lost to control set restrictions	208	69,120	1%	
Number of bonded IOBs	65	640	10%	
Number of LOCed IOBs	60	65	92%	
IOB Flip Flops	21			
Number of BlockRAM/FIFO	21	148	14%	
Number using BlockRAM only	19			
Number using FIFO only	2			
Number of 36k BlockRAM used	17			
Number of 18k BlockRAM used	4			
Number of 18k FIFO used	2			
Total Memory used (KB)	720	5,328	13%	
Number of BUFG/BUFGCTRLs	5	32	15%	
Number used as BUFGs	5			
Number of IDELAYCTRLs	2	22	9%	
Number of PLL_ADVs	2	6	33%	
Number of TEMACs	1	2	50%	
Average Fanout of Non-Clock Nets	6.47			

Matrix Multiplication GF(2): Results

- For multiplication of two 1024 bit x 1024 bit matrices, total execution time= 0.33 sec when we blocks are re-used and about 3.30 sec when no reuse.
- 203 clock cycles at 167MHz .
- Block multiplication implemented in naïve way. Can be optimized further but Ethernet communication is major bottleneck

Matrix Multiplication GF(2): Comparison

- M4RI [17] probably the fastest library for arithmetic over GF(2).
- Uses “Method of four Russians” and look-up-table based tricks to efficiently compute GF2 matrix product.
- Compute time for 10,000x10,000 matrix product on i7 processor = 1.504 seconds. $O(n^{\log\_2(7)})$ algorithm.
- However, its much slower for GF(2^e) e>1 computations. CPU-FPGA likely to be faster in this case

Other Applications using SIRC

- Used SPIRAL [21] core generator for 32-bit fixed-point 8-point DFT.
- 8-point input available in BRAM of FPGA. FPGA computes the DFT and sends the result to CPU when asked.
- Takes ~0.56 sec for execution. Again communication is the major bottleneck.

Discussion

- Communication is the major bottleneck.
- At 450 Mbps, ~75 usec required for transfer of 33 blocks of 32 bit x 32 bit size. => for 2*32*32 such transfers, we would require ~153 msec.
- In addition to ethernet transmission delay, ~65usec additional overhead incurred by SIRC protocol for each transfer. ~133 msec for 2*32*32 transfers.
- ~286msec out of ~330 msec spent in communication. Major bottleneck!

Future Work

- Use SIRC to generate multiple API interfaces to exploit parallelism and streaming-style execution with overlapped I/O.
- Study and exploration of open-source profiling and partitioning tools.
- Work towards a completely automated C++-to-CPU-FPGA tool flow by integrating SIRC with open-source profiling and HLS tools.

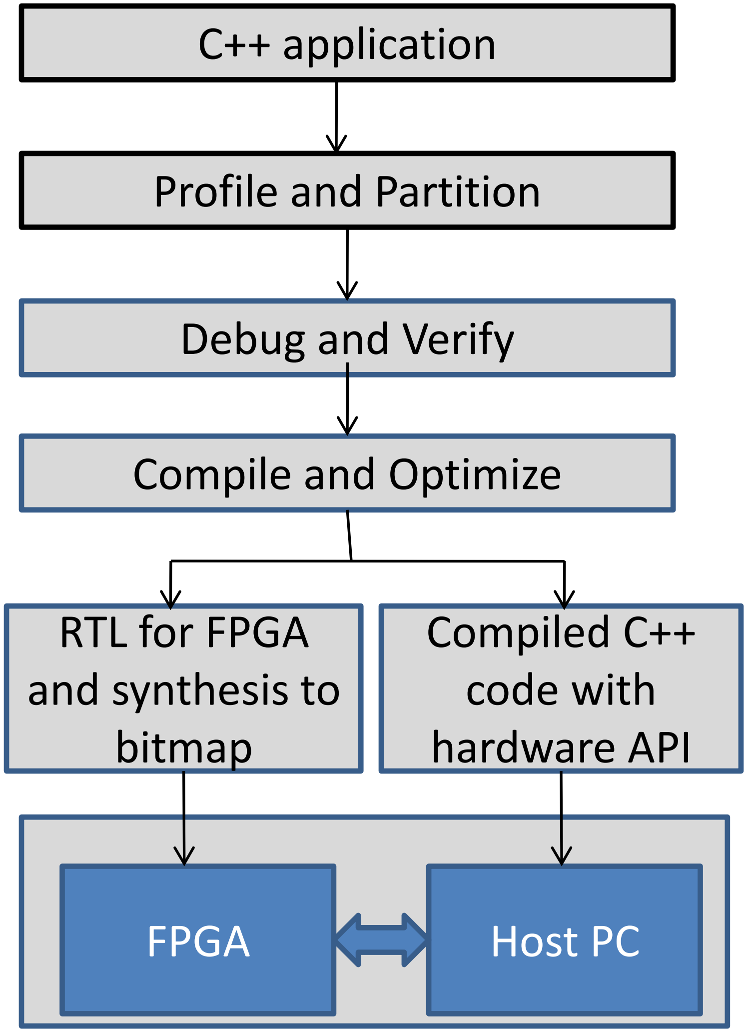


Figure 5. Automated CPU-FPGA tool flow

Future Work

- Explore of benefits of hardware acceleration for other applications and domains:
  - Sparse Matrix-Vector Product. Gaussian elimination
  - LU decomposition/ Cholesky decomposition
  - Coding theory: RS/ LDPC/ Expander decoding
  - Cryptography: DES, Rijndael etc
  - Signal and Image Processing: Wavelet transform, filtering etc
  - Algorithms: Monte Carlo, BlackScholes, PageRank etc
- More detailed study and comparison of performance with benchmarks.
- Build on a strong library foundation for FPGA-based accelerated computing of applications for programmability and reusability.

Limitations

- Communication over Gigabit Ethernet: too slow for data-intensive applications. Communication latency is the major bottleneck.
- SIRC cannot simultaneously read and write to FPGA with single interface.
- High programming costs for low level HDLs. Open source high level synthesis tools [3] [18] not most optimum.
- Separate tool flows and programming for hardware and software. Need for a single programming model.

Limitations

- Cost of reconfiguration too high! Need to minimize reconfiguration time to as small as possible to enable reconfiguration at run-time as well.
- SIRC does not support multi-context reconfiguration.
- Need better tools for programming and analysis of hybrid computing. Need to know where cycles are being spent!

Conclusion

- Literature survey on recent trends in HPC and hybrid CPU-FPGA computing.
- Study and implementation of algorithms on FPGA using high-level synthesis tools such as C-to-Verilog [3] and SPIRAL [21].
- Study and implementation of GF2 32x32 bit matrix multiplication using SIRC for hybrid CPU-FPGA application. The codes can serve as templates for a variety of application we plan to implement in future.
- FPGA has limited resources and memory. Unable to implement highly complicated hardware blocks.

Conclusion

- Extending GF2 multiplication for larger matrices using block multiplication.
- Preparing a roadmap and laying a strong foundation for further research and exploration in this direction.

References

- [1] “Variable SMP – A Multi-Core CPU Architecture for Low Power and High Performance,” pp. 1–16.
- [2] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” *Proceeding of the 38th annual international symposium on Computer architecture - ISCA ’11*, p. 365, 2011.
- [3] “Rotem, Nadav, et al. ‘C-to-Verilog.’ Automating circuit design.[HTML](http://c-toverilog. com) (2010).”
- [4] Wiki, “GF(2).” [Online]. Available: http://en.wikipedia.org/wiki/GF(2).
- [5] Wiki, “Block Multiplication.” [Online]. Available: https://en.wikipedia.org/wiki/Block_matrix.
- [6] Peter Richards and Stephen Weston, “Technology in Banking: a problem in scale and complexity,” 2011.
- [7] T. Trend and T. Reconfigurable, “Dynamically Reconfigurable Processors,” 1998.
- [8] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, “Reconfigurable computing : architectures and design methods,” vol. 152, no. 2, pp. 193–207, 2005.

References

- [9] Z. Peng, “High-Level Synthesis,” pp. 1–24. [ONLINE]: http://www.ida.liu.se/~petel/SysSyn/lect3.frm.pdf
- [10] S. Nematbakhsh, G. Stitt, F. Vahid, S. Nematbakhsh, G. Stitt, and F. Vahid, “The Effect of FPGA Size on Software Speedup from Hardware / Software Partitioning The Effect of FPGA Size on Software Speedup from Hardware / Software Partitioning.”
- [11] P. Greenhalgh, “Big . LITTLE Processing with ARM Cortex ™ -A15 & Cortex-A7,” no. September, pp. 1–8, 2011.
- [12] K. Eguro, “SIRC : An Extensible Reconfigurable Computing Communication API,” pp. 1–4.
- [13] K. Eguro, “Simple Interface for Reconfigurable Computing (SIRC): PC ⍰ Xilinx V5 / V6 Communication,” vol. 2007, pp. 1–33, 2007.
- [14] A. Corporation, “White Paper FPGA Coprocessing Evolution : Sustained Performance Approaches Peak Performance,” no. June, pp. 1–8, 2009.
- [15] G. Chatziparaskevas, A. Brokalakis, and I. Papaefstathiou, “An FPGA-based Parallel Processor for Black-Scholes Option Pricing Using Finite Differences Schemes,” 2012.
- [16] M. Taylor, “Is Dark Silicon Useful ? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse,” 2012.
- [17] “M4RI- Open-source Linear Algebra Library.” [Online]. Available: http://m4ri.sagemath.org/.
- [18] Andrew Canis et. al. 2011. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays* (FPGA '11). ACM, New York, NY, USA, 33-36.

References

- [19] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, “xPilot : A Platform-Based Behavioral Synthesis System.”
- [20] “Pico Computing.” [Online]. Available: http://picocomputing.com/.
- [21] “SPIRAL code generator.” [Online]. Available: http://www.spiral.net/codegenerator.html.
- [22] A. Schopenhauer, G. M. Curve, and M. Age, “THE VON NEUMANN SYNDROME TU Kaiserslautern , http://hartenstein.de Energy Wall , Memory Wall and Education Wall fuel the von Neu- von Neumann syndrome :,” 2010.
- [23] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, “Accelerating Compute-Intensive Applications with GPUs and FPGAs.”

- VN paradigm
- Limitations of VN paradigm
- Reconfigurable computing defining a new paradigm
- Comparison: CPU vs FPGA vs GPU
- Advantage of reconfigurable computing
- Further motivation and ongoing work
- High Level Synthesis
- SIRC
- Protocol and programming
- Contributions of this work
- Implementation Details
- Experimental Study
- Observations and Results
- Discussion
- Future Work
- Limitations
- Conclusion
- References