GEORGE MASON UNIVERSITY

Option Space Exploration Using Distributed Computing for Efficient Benchmarking of FPGA Cryptographic Modules

Benjamin Brewster, Ekawat Homsirikamol, Rajesh Velegalati and Kris Gaj
ECE Department, George Mason University, Fairfax, VA 22030, U.S.A.
email: {bbrewste, ehomsiri, rvelegal, kgaj}@gmu.edu
http://cryptography.gmu.edu

CERG

Motivation and Background

ATHENa is an open-source benchmarking environment aimed at:

- Automated generation of
- Optimized results for
- Multiple hardware platforms.

Distinguishing features of ATHENa:

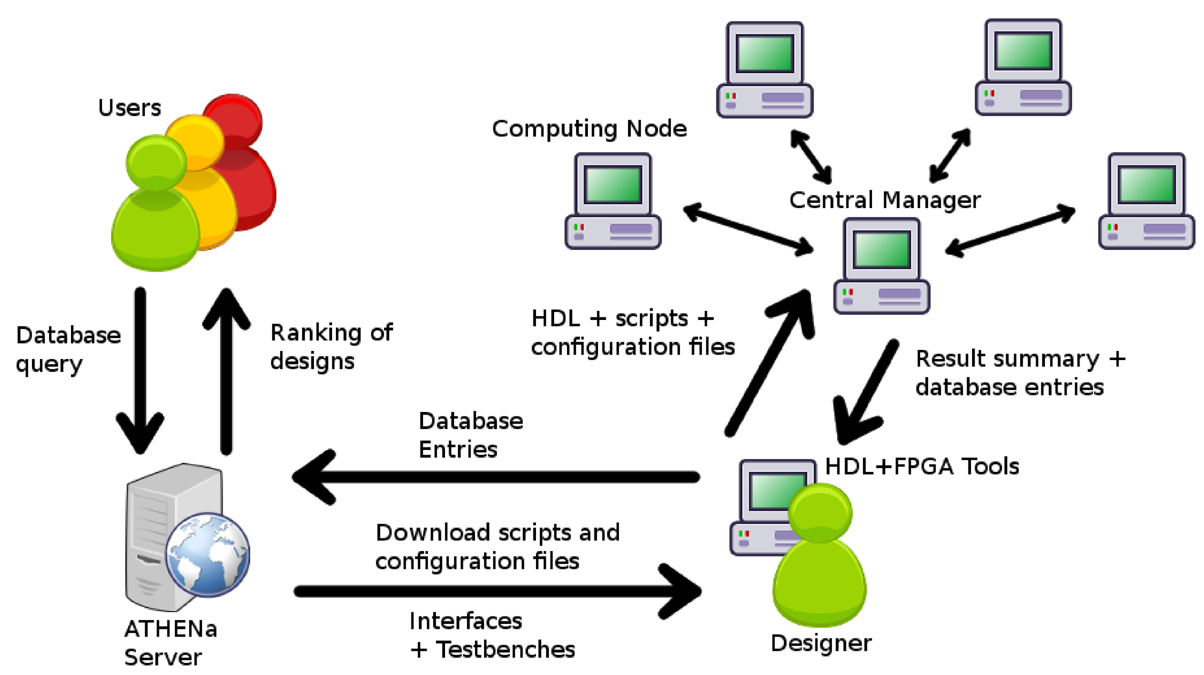
- Support for multiple tools from multiple vendors
- Optimization strategies aimed at the best possible performance
- Extraction and presentation of results
- Seamless integration with the ATHENa database of results

- Flexible toolchain which can support third party tools
- Better utilization of machines via parallel operation of computing nodes
- Save time for users in managing large hardware benchmarking project.

Limitations of the previous version of ATHENa:

- Previous heuristic algorithms used required significant amount of run time
- Unable to utilize parallelism across computing nodes
- Not easy to maintain

Proposed Environment and Improvement



Major Improvements

- Parallel Execution on Multiple Computers
  - Utilize idle resources
  - Increase throughput of benchmarking tasks
  - Decrease benchmarking time
- Optimization Space Exploration
  - Search more options
  - Decrease search time
  - Increase optimization end-performance
- Usability
  - GUI
  - Monitoring and control
  - Benchmark configuration

Optimization Algorithms

- Utilize algorithms inspired by previous research on the programming language compilers
  - Least Effort - LE
  - Most Effort - ME
  - Batch Elimination - BE
  - Iterative Elimination - IE
  - Orthogonal Arrays - OA
- Optimize FPGA-specific algorithms introduced in previous version of ATHENa
  - Frequency Search - FS
  - Placement Search - PS

Least Effort & Most Effort

- Least Effort - minimum execution time, worst results
  - Lazy or naïve optimization
  - Used as a baseline
  - Minimum amount of work needed to optimize
  - Almost never optimal
- Most Effort - maximum execution time, best results
  - Also known as Exhaustive Search
  - Guarantee optimal result
  - Least time-efficient
  - Impractical for more than a handful of options
  - Number of runs needed: 2n, where n is the number of options

Frequency Search & Placement Search

- There are two largest driving factors in performance for cryptographic cores in Xilinx FPGAs
  1. The desired input frequency we wish to achieve - Frequency Search (FS)
  2. A seed value for the tools to begin the placing process - Placement Search (PS)
- **Frequency Search** (FS) attempts to determine the input frequency that yields the highest performance from the design

$$Fin_n = Fout_o * [1 + (.1 * n)], \ n \text{ from 1 to 10}$$

- **Placement Search** (PS) is a very basic search that does an exhaustive search of a subset of possible placement values then refines the search and performs a second exhaustive search on a more granular set of placement options.

Batch Elimination

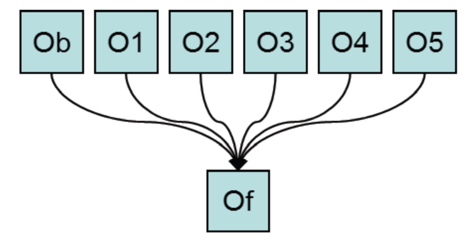
- **Based on:** Z. Pan and R. Eigenmann, Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning, Proc. International Symposium on Code Generation and Optimization, CGO 2006.

Run	Opt.1	Opt.2	Opt.3	Opt.4	RIP
Ob	0	0	0	0	N/A
O_1	1	0	0	0	10%
O_{2-1}	0	1	0	0	20%
O_{2-2}	0	2	0	0	-5%
O_3	0	0	1	0	15%
O_4	0	0	0	1	8%
O_f	**2**	0	**1**	**1**	*N/A*

*Notation: RIP - Relative Improvement Percentage

$$RIP(O_i) = \frac{P(O_i{=}1) - P(O_i{=}0)}{P(O_i{=}0)} \times 100\%$$

$$RIP_B(O_i = 1) = \frac{P(O_i{=}1) - P_B}{P_B} \times 100\%$$

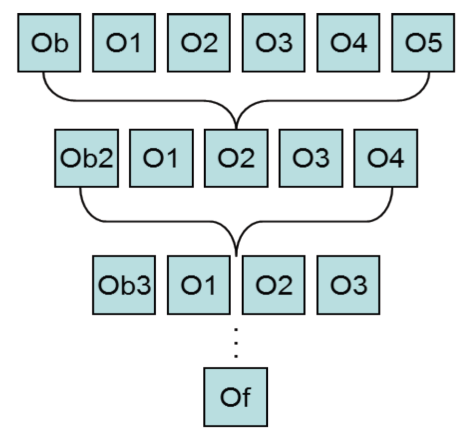


- Ob - Baseline with all options off
- O_i - Option i on, i=1..n
- O_{i-j} - i option with j state
  - if more than one state is available
- O_f - Final options

- **Number of runs:** $n + 2$
- **Number of run levels:** 2

Iterative Elimination

- **Based on:** Z. Pan and R. Eigenmann, Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning, Proc. International Symposium on Code Generation and Optimization, CGO 2006.
- Iterative Elimination takes into account the interaction of optimization options into consideration
- Increases algorithm time complexity



- Ob - Baseline option
- O_i - Option i on, i=1..n
- O_{i-j} - i option with j state
  - if more than one state is available
- O_f - Final options

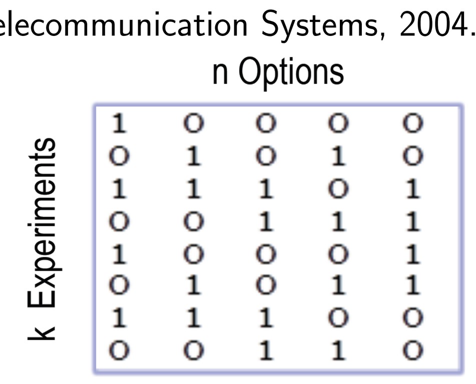
- **Number of runs:** $[n * (n/2)] + (n/2)$
- **Number of run levels:** n

Run	Opt. 1	Opt. 2	Opt. 3	Opt. 4	RIP
Ob_1	0	0	0	0	N/A
O_{1-1}	1	0	0	0	10%
O_{1-2}	2	0	0	0	20%
O_2	0	1	0	0	-5%
O_3	0	0	1	0	15%
O_4	0	0	0	1	8%
$Ob_2 = O_{1-2}$	*2*	0	0	0	+20%*
O'_2	*2*	1	0	0	10%
O'_3	*2*	0	1	0	-3%
O'_4	*2*	0	0	1	4%
$Ob_3 = O'_2$	*2*	*1*	0	0	+32%*
O''_3	*2*	*1*	1	0	-2%
O''_4	*2*	*1*	0	1	-7%
$O_f = Ob_3$	**2**	**1**	0	0	**+32%***

*with respect to Ob_1
*Notation: RIP - Relative Improvement Percentage

Orthogonal Arrays

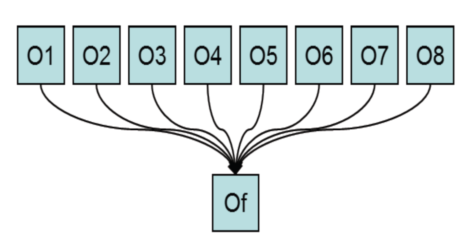
- **Based on:** R.P.J. Pinkers, P.M.W Knijnenburg, M. Haneda, and H.A.G. Wijshoff, Statistical Selection of Compiler Options, 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2004.



- k x n matrix where
  - rows → settings used for each experiment
  - columns → optimization options
- The matrix is filled with 1's and 0's to represent whether or not a specified option is on or off
- Any two arbitrary columns contain the patterns: 00, 01, 10, 11

- The algorithm guarantees that half of the experiments will be conducted with an options O_i on and the other half with O_i off
- For arbitrary two options O_i and O_j there are exactly $k/4$ experiments per each possible setting of these two options

Run	Opt.1	Opt.2	Opt.3	Opt.4	Opt.5
O_1	1	0	0	0	0
O_2	0	1	0	1	0
O_3	1	1	1	0	1
O_4	0	0	1	1	1
O_5	1	0	0	0	1
O_6	0	1	0	1	1
O_7	1	1	1	0	0
O_8	0	0	1	1	0
RIP	+	+	-	-	+
O_f	1	1	0	0	1



- **Number of runs:** $k + 1$
- **Number of run levels:** 2

$$RIP(O_i) = \frac{\sum P(O_i{=}1) - \sum P(O_i{=}0)}{\sum P(O_i{=}0)}$$

Experiments

- **Codes:** 2 SHA-3 candidate algorithms: BLAKE and JH
- **FPGA families:** Spartan 3 and Virtex 6
- **Version of tools:** Xilinx ISE v.13.1
- **Hosts:** Two eight core Linux workstations = total of 16 execute nodes
- **Optimization Target:** Throughput/Area Ratio
- **Experiment 1**
  - Limited search to 5 options
  - Determine ability of Batch Elimination, Iterative Elimination and Orthogonal Array to optimize results
- **Experiment 2**
  - Used expanded 9 option set and optimization algorithms chaining
  - Determine whether further improvement can be achieved if more options and algorithms chaining are used

Results

Experiment 1 Results

Spartan 3

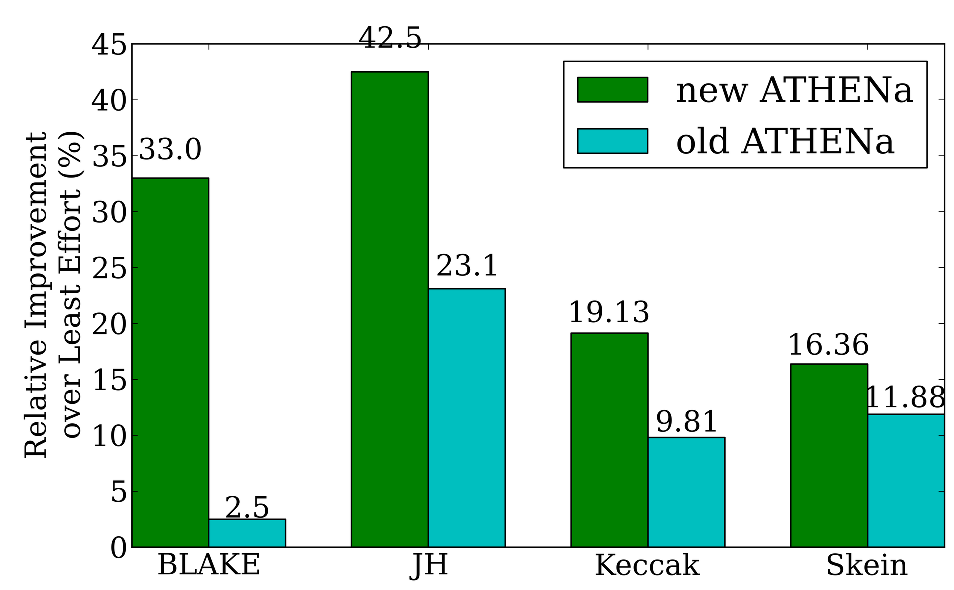
	Above Least Effort (%)			Below Most Effort (%)		
	BE	**IE**	**OA**	**BE**	**IE**	**OA**
JH	5.3	16.0	15.5	-9.8	-0.7	-1.1
BLAKE	7.9	33.0	-3.0	-18.9	0	-27.1
Skein	3.3	5.9	-1.9	-12.8	-10.6	-17.1
Keccak	-1.3	10.8	8.5	-10.9	0	-2.1
Average %inc	3.8	16.4	4.7	-13.1	-2.8	-11.9
Median %inc	4.3	13.4	3.2	-11.8	-0.3	-9.6

Virtex 6

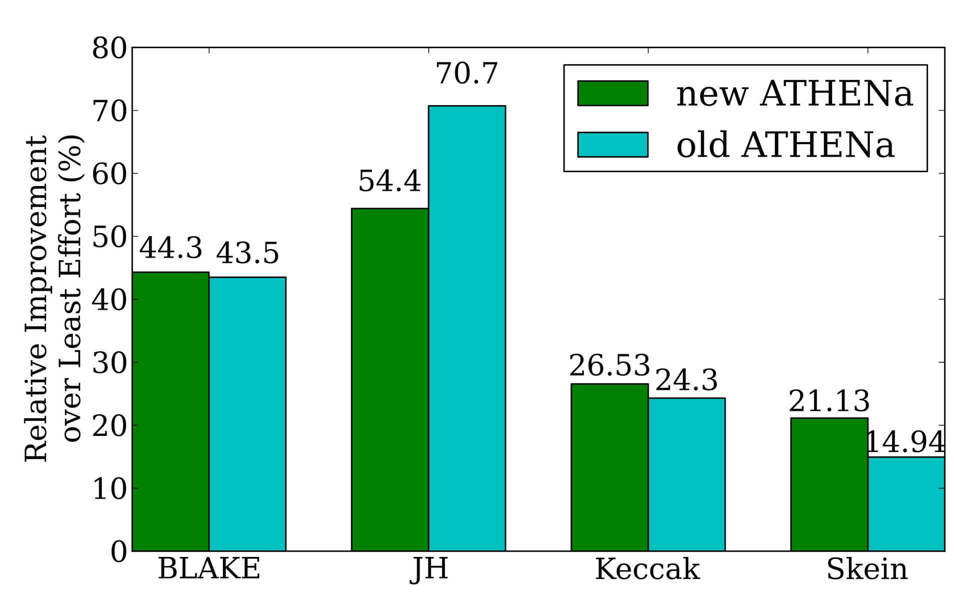
	Above Least Effort (%)			Below Most Effort (%)		
	BE	**IE**	**OA**	**BE**	**IE**	**OA**
JH	8.6	13.5	13.5	-4.3	0	0
BLAKE	26.4	36.4	26.5	-7.3	0	-7.3
Skein	-2.6	9.4	7.2	-11	0	-2.0
Keccak	-2.6	1.1	-3.7	-8.5	-5.1	-9.6
Average %inc	7.5	15.1	10.9	-7.8	-1.3	-4.7
Median %inc	3	11.4	10.3	-7.9	0	-4.6

Experiment 2 Results

Spartan 3



Virtex 6



Conclusion

- Distributed architecture and parallelization increase throughput of benchmarking tasks
- Parallelization extended beyond core count of a single machine
- More efficient use of resources
- Greater tool flexibility
- More heuristic search options
- Increases number of effectively searched options
- Iterative Elimination is a viable alternative to Most Effort optimization with larger options sets
- Optimization algorithm chaining yields results that outperform previous version of ATHENa and Xilinx PlanAhead.