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

Benjamin Brewster, Ekawat Homsirikamol, Rajesh Velegalati, Kris Gaj

ECE Department, George Mason University
4400 University Drive, Fairfax, VA 22030, USA
email: {bbrewste, ehomsiri, rvelegal, kgaj}@gmu.edu

***Abstract*—Benchmarking of digital designs targeting FPGAs is a time intensive and challenging process. Benchmarking results depend on a myriad of variables beyond the properties inherent to the designs being evaluated, encompassing the tools, tool options, FPGA families, and languages used. In this paper we will be discussing enhancements made to the ATHENa benchmarking tool to utilize distributed computing as well as option space exploration techniques to increase the efficiency of the pre-existing process. The capabilities of our environment are demonstrated using two example designs from the SHA-3 cryptographic hash function competition, BLAKE and JH.**

I. Introduction

The open-source benchmarking environment called ATHENa [1], [2] (Automated Tool for Hardware EvaluatioN) aims to address the difficulties associated with the fair comparison of digital systems designed and modeled using hardware description languages and implemented on FPGAs. These comparisons require thorough benchmarking of the digital systems to determine the optimal set of design tool options to use given an FPGA family and device target that achieves the highest performance relative to the selected evaluation metric. Determining the optimal set of tool options is a task that requires searching the entire option space of the tools used for FPGA implementation. This task becomes infeasible for large tool option sets provided by modern FPGA implementation tools.

Option space exploration is intended to determine a near optimal set of options using optimization algorithms to reduce the search space and increase the efficiency of the comparisons done with ATHENa. A distributed computing environment is used to extract the parallelism from these benchmarking tasks and increase the throughput of the benchmarking process. As ATHENa has gained traction as a tool for comparing competing algorithms, particularly in the cryptographic space, efficiency in comparison is an issue that needs to be addressed.

II. Previous Work

There are other systems that aim to accomplish similar goals to the ones being accomplished with enhanced ATHENa. eBACS [3] (ECRYPT Benchmarking of Cryptographic Systems) is a system similar to ATHENa that focuses on the benchmarking of cryptographic primitives in software. Different processor architectures are evaluated as well as various compiler options.

Xilinx has developed an integrated design environment called PlanAhead that contains many similar features to ATHENa. The PlanAhead software allows multiple synthesis and implementation attempts using different options and constraints. The synthesis and implementation attempts can be queued to launch sequentially or simultaneously with multiprocessor machines using the Xilinx synthesis and implementation software. PlanAhead also contains predefined option sets for benchmarking purposes. PlanAhead while similar to ATHENa does not support any other vendor device or tool chains limiting its use to Xilinx devices and tools alone.

Design Space Explorer is a tool similar to Xilinx PlanAhead, but integrated into the Quartus tools for use with Altera devices. It is meant to be an easy-to-use design optimization utility that can assist in optimizing a design in terms of power, area, and speed. A predefined set of Quartus II options are searched to determine optimal settings for a particular design.

The main advantages of ATHENa over the aforementioned programs are the multi-vendor device and tool chain support, optimization strategies aimed at maximum performance as opposed to design closure, and the automated extraction of results that integrates seamlessly with a results database [2].

III. Environment

One of the largest drawbacks to the previous version of ATHENa is the inability of the system to efficiently process large benchmarking and comparison workloads. It was determined that a way to extract the parallelism that exists in these jobs and distribute them across multiple compute nodes would greatly increase the effectiveness of the system. In conjunction with this, new option space exploration algorithms were designed to limit the amount of processing required by the benchmarking jobs to effectively reach the optimal or near optimal set of options for a specific design. By combining

978-1-4673-2845-6/12/$31.00

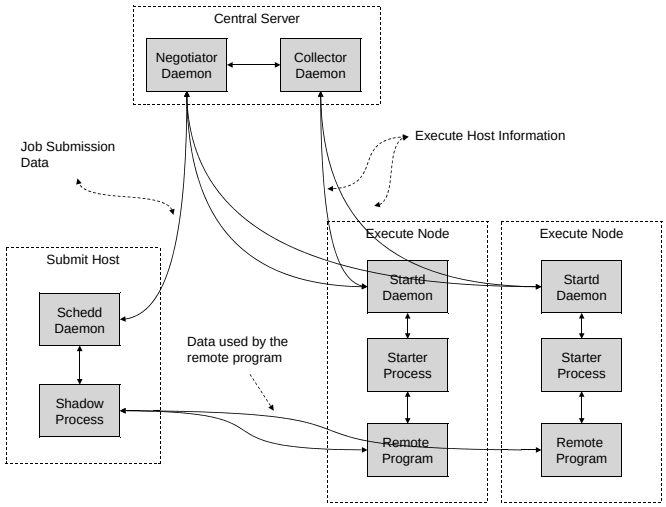

Fig. 1. Distributed Athena Architecture

these new features ATHENa becomes much more powerful and practical to use.

Enhanced ATHENa was designed from the beginning to utilize parallelism as much as possible. The tool was developed in Python, and runs on Condor, a workload management system for compute intensive jobs [4]. The benefits of this framework are the ability to work across tools from different vendors, more customizable tool-chain, and the open-source nature of the project.

The system can be divided into three distinct parts, 1) submit host, used to submit benchmarking jobs, 2) execute hosts, used to process jobs in a distributed fashion, and 3) central server, used to schedule jobs based on available resources. Fig. 1 shows a top level view of enhanced ATHENa's architecture.

The client application consists of a GUI for submitting and monitoring jobs. Information regarding each job is displayed together with controls to pause, resume, and cancel existing jobs. Behind the GUI is an application that extracts the relevant information from the job configurations, and then sends the job to the server side application via a batch system. The application will break a user submitted job into independent subtasks for processing in parallel, if possible, and will send these subtasks to the batch system separately.

The server side application, running on execute nodes, is very similar to the original ATHENa in design. It receives sets of configuration options and executes the synthesis and implementation tools with those options. Results are generated by the application, and the batch system is notified of job completion. Finally, the remote program collects these results for transporting back to the machine that the user submitted the job from.

The central server collects information about resources available on execute nodes, and assigns submitted jobs to these nodes accordingly. In a common scenario, the central server runs on the same machine as the submit host.

IV. Optimization Algorithms

A. Least Effort (LE)

Least effort optimization is not really an optimization method at all, but simply a method used to compare other algorithms against. Least effort consists of setting all options to their perceived high state. This enables us to compare the actual benefit of searching the option space more thoroughly and establish a baseline for comparison.

B. Most Effort (ME)

Most effort optimization is comprised of an exhaustive search of all option combinations within the option search space. An exhaustive search will yield the optimal option set for the design but places an enormous burden on resources by requiring that all option combinations be tested. An exhaustive search requires 2^n runs to test the entire option search space for *on/off* options, where n is the number of options being investigated.

C. Batch Elimination (BE)

The Batch Elimination algorithm proposed here is adapted from previous research in compiler option selection [5]. Batch elimination is an algorithm that aims to approximate an exhaustive search of the optimization options in a much more efficient manner. The benefit of this algorithm is that it requires only $n+2$ optimization combinations to complete, (with n being the number of options), a significant savings compared to exhaustive search. While significantly faster than an exhaustive search, there is no guarantee that an optimal combination will be discovered.

The algorithm itself is quite simple in design. The effect of one option, O_i, is represented by a metric called Relative Improvement Percentage, $RIP(O_i)$ in Eq. 1, which is the difference in performance relative to a chosen metric of the benchmark with and without option O_i. $P(O_i = 0)$ is the performance measured when the option is off, and $P(O_i = 1)$ is the performance measured when the option is on.

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

The baseline used for comparison is the performance of the design with all options switched off. The relative improvement of an option as compared to the baseline is defined in Eq. 2.

TABLE I
Batch Elimination Example

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_3	0	1	0	0	-5%
O_4	0	0	1	0	15%
O_5	0	0	0	1	8%
O_f	***2***	0	***1***	***1***	*N/A*

$$RIP_B(O_i = 1) = \frac{P(O_i = 1) - P_B}{P_B} \times 100\% \qquad (2)$$

If $RIP_B(O_i = 1) > 0$, i.e., the option O_i has a positive effect on performance, the Batch elimination algorithm enables this option in the final option set. The pseudocode for the algorithm is as follows:

1) Run the FPGA synthesis and implementation tools using the baseline option set (all options set to *off*) to generate P_B.
2) For each option being investigated O_i , switch the option on and run the synthesis and implementation tools to generate $P(O_i = 1)$. Determine the $RIP_B(O_i = 1)$.
3) Enable all options with positive RIP_B. Run the synthesis and implementation tools with this tuned option set to determine the final performance metric.

The options are not limited to binary *on/off* option types as shown above and can be n-ary in nature. In the case of an n-ary option performance is calculated for each option state, $P(O_i = 1)$, $P(O_i = 2)$, ..., $P(O_i = n)$ and the option state with the greatest RIP is used in the final option set. An example of a batch elimination with 4 investigated options is shown in Table I, where Ob_i - baseline i, O_{i-j} - i option with j state (if more than one state is available), and O_f - final run.

D. Iterative Elimination (IE)

Iterative elimination is a modification to the previously defined batch elimination that takes into account the interaction of optimizations into consideration. The price that is paid to deal with interactions is an increase in algorithm time complexity. Iterative elimination performs the RIP calculation described in batch elimination for each option, but then only sets the most beneficial option at a time, creating a new baseline and continuing the process iteratively until options no longer yield a positive RIP relative to the last baseline. An example of Iterative Elimination with 4 options is shown in Table II.

TABLE II
ITERATIVE ELIMINATION EXAMPLE

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

Iterative elimination requires at most $[n * (n/2)] + (n/2)$ runs to complete while on average it can be much less. The following pseudocode describes the algorithm in more detail.

1) Run the FPGA synthesis and implementation tools using the baseline option set (all options set to *off*) to generate PB
2) For each option being investigated O_i , switch the option on and run the synthesis and implementation tools to generate $P(O_i = 1)$. Determine the $RIP_b(O_i = 1)$.
3) Find the option Oi , that has the greatest RIP_b. Create a new baseline with this option set to *on* instead of *off*.
4) Repeat the previous steps until there is no positive RIP_b for any option. The final baseline will represent the fully optimized option set.

E. Orthogonal Arrays (OA)

Orthogonal arrays have been used as an efficient means to design experiments in many fields of engineering. Recent research has been done using orthogonal arrays as a means to choose gcc compiler options, an application that mirrors ours [6], [7]. In our experiment, we have n factors being considered, creating an optimization space with 2^n settings. This is known as a full factorial design. As n increases, it is not possible to test the entire search space to determine the optimal option combination. Orthogonal arrays are a means to create an experiment that is called a fractional factorial design, allowing meaningful information to be determined about option combinations while searching only a fraction of the full factorial design optimization space.

An Orthogonal Array is an $k \times n$ matrix where the columns represent optimization options and the rows represent the settings used for each experiment. The matrix is filled with 1s and 0s to represent whether or not a specified option is on or off. The property that makes Orthogonal Arrays useful for creating fraction factorial designs is that any two arbitrary columns contain the patterns 00, 01, 10, 11 equally often. An Orthogonal array of this type is said to be of strength 2, referring to the two column patterns. A strength 2 array allows the discovery of main effects for options, similar to the non-interacting option detection that is used in batch elimination. The effect of options interacting among one another can be analyzed with greater strength Orthogonal Arrays and should achieve results similar to iterative elimination. An example of an Orthogonal Array of strength 2 is shown in Fig. III with eight rows corresponding to eight separate experiments, or sets of options. The array contains five columns, representing a search space of five options.

Orthogonal Arrays allow us to view performance when an option is switched on and off in an arbitrary context of other options. This algorithm also guarantees that half of the experiments will be conducted with an option O_i *on* and half with option O_i *off*. The algorithm provides another guarantee, for an arbitrary option O_j, there are N/4 experiments where O_i is *on* and O_j is *off*, and N/4 experiments where O_i is *on* and O_j is *on*. The same holds true for when O_i is *off*. These properties

allow the Orthogonal Array algorithm to show main effects of options without resorting to a full factorial design. To properly utilize Orthogonal Arrays to conduct option space optimization we need to define a metric called relative improvement, similar to that calculated for the prior two algorithms.

$$RIP(O_i) = \frac{\sum P(O_i = 1) - \sum P(O_i = 0)}{\sum P(O_i = 0)} \quad (3)$$

where $P(O_i = 1)$ is the performance when the option O_i is enabled and $P(O_i = 0)$ is the performance when option O_i is disabled. The relative improvement for each option is calculated after running all of the experiments defined by the Orthogonal Array. Any options that yield a positive relative improvement are enabled in the final set while those yielding no improvement or negative improvement are disabled.

The algorithm as designed for ATHENa only supports binary options. Options that have more than two states are not considered and require more complicated Orthogonal Arrays to support. For our purposes a strength 2 array is used, so only main effects are taken into account leading to a less optimal final solution. Higher strength arrays are more complicated and so a trade-off was made for increased simplicity over better results. We hope to show that in most cases main effects are sufficient and further complexity yields diminishing returns for our specific application.

F. Other Algorithms

Along with the above mentioned algorithms for choosing the best option sets, there also exist two other algorithms that have been adapted from the original ATHENa. They are Frequency Search and Placement Search. Two of the largest driving factors in performance for cryptographic cores in FPGAs are the desired input frequency you wish to achieve and a seed value for the tools to begin the placing process. Frequency Search (FS) attempts to determine the input frequency that yields the highest performance from the design. Frequency Search is accomplished as follows:

1) An initial input frequency Fin_0 is chosen a starting point for the Frequency Search. An initial option set is also used as input. All options are set to the tool defaults.
2) $Fout_0$ is the frequency generated from running the tools with an input frequency of Fin_0.
3) A set of values used as input frequencies for the next stage Fin_n are determined.

TABLE III
ORTHOGONAL ARRAY EXAMPLE

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

$$Fin_n = Fout_o * [1 + (.1 * n)],\ n\ from\ 1\ to\ 10 \quad (4)$$

4) $P(Fin)_{max}$ is determined from the set of results generated with the input frequency set Fin_n. Fin is used as the input to the next stage of the Frequency Search.
5) $Fin_n = Fin * [1 + (.03 * n)]$ is determined for n = -3, -2, -1, 1, 2, 3
6) $P(Fin)_{max}$ is determined again for the new input set and a new Fin is used as input to the final stage of Frequency Search.
7) $Fin_n = Fin * [1 + (.01 * n)]$ is determined for n = -2, -1, 1, 2
8) After this set of runs has completed the input frequency that lead to the best overall performance is selected.

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. For example, placement seed is specified as an integer from 1-100 in the Xilinx tools. The initial exhaustive search searches 1,10,20,30,40,50,60,70,80,90,100 as the value for the placement option. The best result of these runs is picked and a second set of placement values is searched centered on this best value. The new values are +/-5,+/-4,+/-3,+/- 2,+/-1 from this value. After running the second set, the value for placement that lead to the greatest resultant performance is the result of the search.

V. EXPERIMENTS AND RESULTS

To demonstrate the effectiveness of the enhanced ATHENa environment as well as the new optimization algorithms, a set of experiments were conducted. The experiments used two proposed algorithms from the SHA-3 cryptographic hash algorithm competition [9], JH and BLAKE and targeted two Xilinx devices, a Spartan 3 device, xc3s1000fg676-5, and a Virtex 6 device, xc6vlx75tff784-3. The implementation details of these algorithms can be found on the ATHENa website and are described in more detail in [10]. Algorithms were implemented using the newly developed ATHENa API.

In addition to the device and algorithms, different option sets were used to illustrate different properties of the system. A small constrained option set including only 5 options was chosen for both the Spartan 3 and Virtex 6 experiments. This option set was used to show how effectively the option space exploration algorithms could determine an optimal option set. An expanded option set of 9 options was chosen for each device to show how well the algorithms could trim the search space, and how the enhanced ATHENa environment performed with respect to the old ATHENa environment and the Xilinx PlanAhead tool. The option sets are slightly different for the Spartan 3 and Virtex 6 devices because of what options are supported for these devices.

Xilinx PlanAhead uses a predetermined set of settings for different strategies. For our experiments, a design is implemented using all available strategies (54 total) with the best

TABLE IV
BLAKE AND JH EXPERIMENTAL RESULTS

	Algorithm	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs (Run-Time)	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs (Run-Time)
		Spartan 3					Virtex 6				
BLAKE /4(h)/4(v)											
Experiment 1	LE	641	150	0.234	0	1 (1)	199	239	1.656	0	1 (1)
	ME	467	146	0.312	33	32 (1)	145	328	2.259	36.4	32 (1)
	BE	502	127	0.253	7.9	7(3)	158	331	2.093	26.4	7(3)
	IE	467	146	0.312	33	13(4)	145	328	2.259	36.4	16(6)
	OA	614	148	0.241	3	9(2)	211	354	1.676	1.2	9(2)
Experiment 2	LE9	641	150	0.234	0	1(1)	212	346	1.631	-1.5	1(1)
	BE9	502	127	0.253	7.9	11(3)	187	356	1.906	15.1	11(3)
	IE9	467	146	0.312	33	18(3)	145	328	2.259	36.4	36(6)
	FS	650	159	0.245	4.4	21(4)	184	433	2.355	42.2	21(4)
	FS=>BE9	485	126	0.26	10.8	32(6)	211	433	2.051	23.9	32(6)
	FS=>IE9	463	129	0.278	18.6	40(7)	176	432	2.457	48.4	66(13)
	FS=>BE9=>PL	459	136	0.297	26.8	53(8)	203	443	2.18	31.7	53(8)
	FS=>IE9=>PL	463	138	0.297	26.9	61(9)	176	432	2.457	48.4	87(15)
	GMU-1	641	154	0.24	2.5	57(19)	182	432	2.376	43.5	57(19)
	XPA	599	137	0.228	-2.7	54(1)	207	422	2.039	23.1	54(1)
JH x2											
Experiment 1	LE	4084	1530	0.375	0	1(1)	1170	4370	3.735	0	1(1)
	ME	3525	1532	0.435	16	32(1)	1024	4342	4.241	13.5	32(1)
	BE	3621	1428	0.394	5.3	7(3)	1080	4379	4.055	8.6	7(3)
	IE	3525	1532	0.435	16	10(3)	1024	4342	4.241	13.5	15(5)
	OA	3665	1409	0.385	2.6	9(2)	1459	4386	3.006	-19.5	9(2)
Experiment 2	LE9	3471	1543	0.444	18.6	1(1)	1487	4368	2.937	-21.3	1(1)
	BE9	3471	1449	0.417	11.4	11(3)	1080	4379	4.055	8.6	11(3)
	IE9	3383	1545	0.457	21.9	40(7)	1024	4342	4.241	13.5	36(6)
	FS	3913	1842	0.471	25.7	21(4)	911	5325	5.845	56.5	21(4)
	FS=>BE9	3612	1845	0.511	36.4	32(6)	970	5525	5.696	52.5	32(6)
	FS=>IE9	3607	1844	0.511	36.5	51(8)	905	5344	5.905	58.1	63(11)
	FS=>BE9=>PL	3396	1845	0.543	45.1	53(8)	976	5667	5.807	55.5	53(8)
	FS=>IE9=>PL	3471	1846	0.532	42	72(10)	884	5551	6.279	68.1	84(13)
	GMU-1	3913	1804	0.461	23.1	56(19)	849	5412	6.375	70.7	68(23)
	XPA	3690	1671	0.453	20.9	54(1)	1024	4182	4.084	8.2	54(1)

*Notation: LE - Least Effort, ME - Most Effort, BE - Batch Elimination, IE - Iterative Elimination, OA - Orthogonal Array, XPA - Xilinx PlanAhead, GMU-1 - GMU Optimization on the old ATHENa, FS - Frequency Search, PS - Placement Search, FS=>BE9 - Frequency Search followed by Batch Elimination, FS=>BE9=>PL - Frequency Search and Batch Elimination followed by Placement Search, FS=>IE9=>PL - Frequency Search and Iterative Elimination followed by Placement Search. The 9 following the BE and IE abbreviations represents 9 option set. By default, 5 option set is selected. See [8] for detailed list of all option sets.

result selected. The results of all test cases for the experiments are tabulated in Table IV. The results are categorized by design and target device. The Run-time value shown in Table IV indicates the number of runs required to compute an optimization algorithm under the assumption that there are unlimited number of processors working in parallel. As such, this number represents the level of dependencies in the investigated algorithms for that particular design.

A. Experiment 1

The first experiment is used to illustrate how effective the option space exploration algorithms are at reaching an optimal option set. The option set is constrained to a set of 5 options to limit the exhaustive search space for this experiment. Five optimization algorithms are used as a basis for comparison. The first is Least Effort optimization which is used as a baseline. The second optimization algorithm, the Most Effort optimization is used to determine if the newly proposed algorithms reach an optimal solution and if not are they close to optimal. The final three runs consist of the newly proposed algorithms Batch Elimination, Iterative Elimination and Orthogonal Array Optimization.

This initial experiment demonstrates some of the advantages and issues of the various algorithms. In the majority of the cases Iterative Elimination reached an optimal solution with the same performance as an exhaustive search. Batch Elimination failed to reach the optimal solution in any of the test cases, but outperformed the Least Effort optimization for all cases. The failure of Batch Elimination algorithm in these cases can be attributed to the interaction effect between the options.

The Orthogonal Array optimization algorithm was much less successful, never reaching the optimal solution and underperforming the Least Effort optimization in the case of the JH on Virtex 6 device. This failure in performance for the Orthogonal Array Optimization can be attributed to a few key issues. First, the array was of strength two, a higher strength array would take more interaction effects of the options into account but also increase the number of runs. Second, the array itself and the way it was implemented only had 8 distinct

combinations. A larger array could be constructed of strength 2 that might also yield better performance at a higher run number cost. Iterative Elimination leads to the best performance at the highest run number cost. Batch Elimination proved to be a good compromise between optimization performance and total run time.

B. Experiment 2

Experiment 2 aims to show the ability of the new option space exploration algorithms to drastically reduce the search space, when compared to Most Effort, while still achieving good optimization performance. In this experiment, an expanded option set of 9 options is utilized. Furthermore, Frequency Search and Placement Search are introduced to show the best performance achievable by the enhanced ATHENa optimization algorithms. These results are then compared to results generated by the previous ATHENa GMU_Optimization_1 (GMU-1) algorithm, and the Xilinx PlanAhead tool. In this experiment, Batch Elimination, Iterative Elimination and Least Effort optimization are used on the expanded option set. Frequency Search is then run with the options utilized by the Least Effort Optimization. The best performing input frequency determined by the Frequency Search is then used as the input frequency for Batch Elimination and Iterative Elimination using the 9 option set.

After these have completed a Placement Search is performed using the options determined by the two previous Batch Elimination and Iterative Elimination runs. The Frequency Search algorithm resulted in a performance increase in all algorithm device combinations. In general, input frequency is the highest driver of overall performance, especially in terms of throughput, as shown in the experiments. This trend was expected as the tool tries to reach timing constraint specified by the user. The rest of the runs show Batch Elimination improving on the performance of Frequency Search for implementations on Spartan 3 but for no implementations on Virtex 6. It can be noted that these are also where the 9 option Batch Elimination struggled to begin with. Following the second Batch Elimination with Placement Search produced better performance in all cases. Iterative Elimination was much more successful at improving performance following the Frequency Search showing performance gains in all cases. In addition Placement Search improved the results in three of the four cases and left the performance unchanged in the remaining case.

VI. Conclusion

This paper presents an enhanced version of ATHENa by developing a new framework allowing more parallelism across computing nodes as opposed to the previous version of limited parallelism within a single computing node. Algorithms are adapted from previous research for hardware implementations. Experiments were performed to demonstrated the ability of enhanced ATHENa and its adapted algorithms to effectively optimize and benchmark designs.

Out of all the algorithms presented, Batch Elimination and Iterative Elimination are effective at pruning the search space while still maintaining the ability to achieve near optimal results. These algorithms coupled with the Frequency Search and Placement Search algorithms can yield the best performance in most cases.

In relation to the older ATHENa environment using the GMU_Optimization_1 optimization algorithm, the enhanced ATHENa outperforms it in each case except for JH when targeting the Virtex 6 device where it only shows a 1.5% disadvantage. The new environment not only outperforms the old one in most cases, but does so while evaluating a larger option space, allowing the potential for larger gains in cases where the options being explored can yield greater benefits. The new ATHENa outperforms its predecessor by 12.9% in terms of performance with an increase of 26.7% on average in the number of total runs but with a reduction of 40% in run-time on average.

In relation to the Xilinx PlanAhead tool, the new ATHENa outperforms the results generated by this tool for each set. It outperforms PlanAhead by a factor of 2.7 with an increase in number of runs of 40.7% on average. These results show that there is a definite advantage to using the enhanced ATHENa environment for optimization and benchmarking tasks.

References

[1] K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster, "ATHENa – Automated Tool for Hardware EvaluatioN: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs," in *20th International Conference on Field Programmable Logic and Applications - FPL 2010*. IEEE, 2010, pp. 414–421, (Winner of the FPL Community Award).

[2] "ATHENa project website," http://cryptography.gmu.edu/athena/, 2012, ATHENa Automated Tool for Hardware EvaluatioN project.

[3] D. J. Bernstein and T. Lange (editors), "eBACS: ECRYPT benchmarking of cryptographic systems," http://bench.cr.yp.to.

[4] "Condor Project Homepage," http://research.cs.wisc.edu/condor/.

[5] Z. Pan and R. Eigenmann, "Fast and effective orchestration of compiler optimizations for automatic performance tuning," in *Proceedings of the International Symposium on Code Generation and Optimization*, ser. CGO '06. Washington, DC, USA: IEEE Computer Society, Mar 2006, pp. 319–332.

[6] R. P. J. Pinkers, P. M. W. Knijnenburg, M. Haneda, and H. A. G. Wijshoff, "Statistical selection of compiler options," in *Proc. IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, ser. MASCOTS '04. Washington, DC, USA: IEEE Computer Society, Oct 2004, pp. 494–501.

[7] M. Haneda, P. M. W. Knijnenburg, and H. A. G. Wijshoff, "Automatic selection of compiler options using non-parametric inferential statistics," in *Proc. 14th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT 2005, Sept 2005, pp. 123–132.

[8] B. Brewster, "Distributed computing and optimization space exploration for fair and efficient benchmarking of cryptographic cores in FPGAs," Master's thesis, George Mason University, Fairfax, Virginia, USA, May 2012.

[9] "SHA-3 Contest," http://csrc.nist.gov/groups/ST/hash/sha3/index.html.

[10] K. Gaj, E. Homsirikamol, M. Rogawski, R. Shahid, and M. U. Sharif, "Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using Xilinx and Altera FPGAs," Cryptology ePrint Archive, available at http://eprint.iacr.org/2012/368.