DISTRIBUTED COMPUTING AND OPTIMIZATION SPACE EXPLORATION FOR FAIR AND EFFICIENT BENCHMARKING OF CRYPTOGRAPHIC CORES IN FPGAS

by

Benjamin Brewster

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

KGaj	Dr. Kris Gaj, Thesis Director
[illegible]	Dr. Jens-Peter Kaps, Committee Member
[illegible]	Dr. Brian Mark, Committee Member
[illegible]	Dr. Andre Manitius, Department Chair
[illegible]	Dr. Lloyd J. Griffiths, Dean, Volgenau School of Engineering
Date: 5/2/12	Spring Semester 2012 George Mason University Fairfax, VA

Distributed Computing and Optimization Space Exploration for Fair and Efficient Benchmarking of Cryptographic Cores in FPGAs

A thesis submitted in partial fulfillment of the requirements for the degree of MasterMaster of Science at George Mason University

by

Benjamin Brewster
Bachelor of Science
West Virginia University, 2005

Director: Kris Gaj, Associate Professor
Department of Computer Engineering

Spring Semester 2012
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to my wonderful wife Megan who put up with the long late nights and endless semesters of work.

ACKNOWLEDGEMENTS

I would like to thank the CERG team that assisted me with all of my research needs and spent many long hours running tests and verifying results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

Automated Tool for Hardware Evaluation	ATHENa
Batch Elimination	BE
Iterative Elimination	IE
Orthogonal Array	OA
Frequency Search	FS
Batch Elimination with nine options	BE9
Iterative Elimination with nine options	IE9
Frequency Search followed by Batch Elimination with nine options	FS=>BE9
Frequency Search followed by Iterative Elimination with nine options	FS=>IE9
FS followed by BE9 followed by Placement Search	FS=>BE9=>PL
FS followed by IE9 followed by Placement Search	FS=>IE9=>PL

ABSTRACT

DISTRIBUTED COMPUTING AND OPTIMIZATION SPACE EXPLORATION FOR FAIR AND EFFICIENT BENCHMARKING OF CRYPTOGRAPHIC CORES IN FPGAS

Benjamin Brewster, MSCpE.

George Mason University, 2012

Thesis Director: Dr. Kris Gaj

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 thesis we will be discussing enhancements made to the ATHENa benchmarking tool to utilize distributed computing as well as optimization space exploration techniques to increase the efficiency of the ATHENa benchmarking process. Capabilities of the environment are demonstrated using four example designs from the SHA-3 cryptographic hashing function competition, BLAKE, JH, Keccak and Skein.

1. INTRODUCTION

The development of new cryptographic standards through competition has necessitated the need for fair and comprehensive benchmarking tools and a methodology for their use that deals with the inherent objective difficulties in these evaluations. Proposed algorithms must be compared in terms of security, cost and implementation flexibility for both their hardware and software implementations. The end goal of ATHENa[1] is to remove the burden of generating high quality benchmarking results from the algorithm designer and generate benchmarking results in a clear unbiased manner for publication and comparison.

Open competition has become the method of choice for adopting new cryptographic standards starting with the AES competition sponsored by NIST in 1997-2000, and followed by the NESSIE and eSTREAM competitions in Europe and the CRYPTREC competition in Japan. The move from traditional standards development to open competition has driven the need for evaluation environments that allow for the fairest possible benchmarking of competing algorithms. Algorithms are typically evaluated based on four criteria, security, performance in software, performance in hardware, and flexibility. While of the utmost concern, the security provided by these algorithms can be incredibly difficult to quantify in the timeline provided by these competitions. After careful examination of an initial pool of algorithm candidates elimination of a few can be done based on inherent security flaws in the algorithms themselves and the remaining candidates can be considered to be equally secure. It has been shown, interestingly enough, that the largest differences tend to emerge in the hardware implementation of the algorithms and this can often serve as the deciding criteria for algorithm adoption in the face of no other clear advantages.

Fair comparison of the hardware efficiency of cryptographic algorithms modeled in Hardware Description Languages and targeted to FPGA platforms is a computationally complex problem that depends on many factors beyond just the algorithm being evaluated and includes targeted architectures, implementation techniques, FPGA families, languages and tools. ATHENa is an evaluation environment that is meant to address these issues with respect to FPGAs and provide a platform to more objectively evaluate and compare new designs. The inherent computational complexity makes the development of a tool that can automate this task as efficiently as possible vitally important to algorithm developers and evaluators alike.

Problem

The main focus of this thesis is the objective evaluation of functionally equivalent algorithms across different metrics implemented in hardware targeted to FPGA platforms focusing on their use in future cryptographic standards. Evaluations of this kind are subject to many pitfalls and difficulties that must be addressed to arrive at a fair comparison which is of interest to a cryptographic community that has become reliant on competition for standards adoption. It is necessary to remove the burden of performing benchmarking and comparisons of algorithms from the algorithm designer and create an

environment that addresses these needs objectively and efficiently. 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 optimization space exploration techniques to increase the efficiency of pre-existing process. Select candidates for the ongoing NIST SHA-3 competition are used as a case study in utilizing the improved ATHENa benchmarking environment. It is shown that through the use of the ATHENa environment the process of objectively comparing candidate algorithms with respect to optimization can be performed much more efficiently and thus aid in their evaluation.

2. BACKGROUND

Cryptographic Standards Competitions

In 1997 NIST announced the desire to adopt a new encryption standard as a replacement to DES. The new standard, AES, was adopted through an open competition in the cryptographic community and competitions of this type have become the preferred method for the development and adoption of new cryptographic algorithms. These competitions require the competing algorithms to be scrutinized and compared in the finest detail. The algorithms must pass rigorous cryptanalysis, hardware and software benchmarking and implementation examination. The amount of work that goes into analysis of these types is incredible and time consuming.

Previous Competitions

Apart for the previously mentioned AES competition, open competition has been used successfully in other cases. NESSIE was a competition in Europe that was aimed at identifying different classes of secure cryptographic primitives. The competition included block ciphers, public-key encryption, MAC algorithms and cryptographic hash functions, digital signature algorithms, and identification schemes. The NESSIE competition lead directly to the formation of the eSTREAM competition to identify stream ciphers due to the fact that all stream ciphers submitted to NESSIE fell to cryptanalysis. Another related competition was the Japanese competition CRYPTREC which was sponsored by the

Japanese government with intent to find cryptographic primitives suitable for industrial and government use.

SHA-3 Competition

The latest cryptographic competition is the NIST sponsored SHA-3 competition. Announced in 2007 its aim is to develop a replacement to the SHA-2 cryptographic hash algorithm. The competition is in its final round of algorithm evaluation and the five finalist algorithms are all used as an illustration of the improvements made to ATHENa as outlined in this text.

Hardware Benchmarking

Hardware benchmarking is a process that requires a lot of care and effort to accomplish properly. With respect to digital systems targeting FPGAs modeled with hardware description languages, many difficulties can arise. The goal of the ATHENa project is to address these goals by creating an environment that tries to eliminate evaluation pitfalls while benchmarking as much as possible.

Benchmarking Pitfalls

Benchmarking pitfalls, also known as evaluation pitfalls are easier to identify and remedy if the evaluation and benchmarking is done with care. One of the most common benchmarking pitfalls is metric selection. Metric Selection is simply ensuring that the designs being compared are using a fair and common metric. Comparison metrics must be chosen with care and must be common across the algorithm implementations that are being used for comparison. Common comparison metrics that are used by ATHENa are throughput, area, and throughput to area ratio. It is also important that a metric such as area is common between designs because area can be composed of different things such as slices, look-up tables, flip-flops, or BRAMs. Within ATHENa the common area metric for Xilinx devices is slices used while Altera devices rely on comparisons using ALUTs.

Another common pitfall is taking credit for technology improvement as algorithm improvement. In FPGAs this can most notably be seen when comparing an algorithm targeting one FPGA family and device to another. The newer devices are higher performing and it is important to only compare algorithms targeting the same device family to account for these technology improvements.

It is also extremely important to ensure that the algorithms being compared and benchmarked against each other have similar functionality. It would be unfair to compare two algorithms with different functionality as proof that one algorithm was superior to the other. It must be ensured that the two algorithms have the same intent and functionality before a meaningful comparison can be made.

Optimization target is another issue that is very similar in nature to the metric pitfall described earlier but relates more directly to targeted optimization than the metric itself. Optimization targets are goals that a system is trying to achieve relative to a given metric. For example if my optimization target was speed I would optimize my design and especially the implementation to achieve the highest throughput. If this was the case it would be unfair for me to compare the speed of my algorithm to that of an algorithm with an area optimization target. It is also essential that the correct numbers are compared. This is especially true in the field of FPGAs where there are many steps to the design implementation and performance metrics could be generated at many of the intermediate steps. It is important that the final placed and routed designs only be compared as these are the true numbers that would manifest themselves on a device. You must ensure that you are not comparing synthesis results to final timing results for instance.

These pitfalls are quite prevalent in many algorithm comparisons and arise for various reasons from simple mistakes to purposefully misleading results. ATHENa has functionality that helps combat these pitfalls and offer a fairer system for comparison and benchmarking.

Orchestration Algorithms

The main area for improvement for ATHENa was identified to be the way that the various option sets offered by the Synthesis and Implementation software were explored and an optimal set chosen for each algorithm. Experimentally it has been shown that changes in the tool options can have an enormous impact on the realized performance of an algorithm with respect to a given metric. The options are numerous enough that it is non-trivial to determine the best option combination provided by the software tools. The goal of a fair comparison of the competing algorithms necessitates the exploration of many option sets for all algorithms to ensure that the true performance is fully discovered. If two algorithms are being compared using handpicked option sets by their creators, it is hard to determine the better design. This is compounded by the fact that option sets vary depending on the specific device, family and vendor chosen as an implementation target.

Previous related research has been undertaken by the computer science community focusing on compiler option selection for optimization of computer programs[4]. While there are differences and caveats that have to be taken into account when comparing work done for software compilers and FPGA synthesis and implementation tools, many parallels exist and previous work can be leveraged to a certain extent. In the section focused on comparing hardware and software benchmarking some of these issues are discussed in greater depth especially pertaining to the methods used by ATHENa as opposed to the eBASH system for software benchmarking. Modern compilers offer many optimization option flags that can be selected to extract greater performance from a given source code. These optimizations offer potentially large performance gains but are largely underutilized given their complexity and the number of options to choose from. The compilers themselves are generally unable to find the best options at compile time and require a user to manually perform feed-back driven analysis of different options sets. The process is further hindered by the fact that certain optimizations can have a negative performance impact on certain source codes so applying a highest optimization level for all options is not always the best approach. The time consuming tedious nature of feed-back driven analysis has led to the development of different orchestration algorithms for the determination of optimization option selection.

Compiler construction-time pruning as proposed in Compiler Optimization-Space Exploration is a method that determines the hot-paths of a program based on profiling data and iteratively constructs new optimization sets for these hot-paths based on unions of previous iterations' sets. The ability to determine hot-paths, or the sections of a program that is executed most often, could be very useful in the domain of FPGAs. Optimizations could be found that benefit individual modules or sections of an FPGA implementation and then optimized separately. While the concept appears promising it is outside the scope of this investigation but could be a subject of future research.

Orthogonal arrays are proposed as a means to statistically analyze profile information generated by an optimized program and measure the main effect of different optimization options in Statistical Selection of compiler Options. The large number of compiler options is too great to perform a full exhaustive search to arrive at the optimal solution, this would require approximately 2^k option sets, where k is the number of options if all options are binary(on/off). The method employed is borrowed from the framework of the Design of Experiments.   Orthogonal Arrays are a matrix of 1's and 0's where each column represents an option and each row is an option set for the compiler, 1 representing the option is on, 0 representing the option is off. An Orthogonal Array has the property that two arbitrary columns contain the patterns 00, 01, 10, 11, equally often. Orthogonal Arrays provide a method for obtaining a fractional factorial design allowing information about a large set of variables be analyzed with much fewer runs than an exhaustive search over the full factorial search space. This approach has many applications in experimental design and testing but the creation of orthogonal arrays that map to the options used by FPGA synthesis and implementation tools is difficult because of the existence of options with more than two levels. This method can be adapted to the binary options provided by the tools and used as a method for FPGA optimization within ATHENa.

Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning also investigates the use of feed-back driven methods to determine the most effective compiler optimizations. Batch Elimination (BE) and Iterative Elimination (IE) are put forth as viable alternatives to an exhaustive search in terms of the optimality of the solution and the time that is required to achieve near optimal results. It is shown that by isolating each option and comparing the isolated option's effect against a baseline that the effect of loosely interacting options can be determined. This is the idea behind batch elimination. To take into account the effect of one option on another iterative elimination can be employed to perform successive batch eliminations in a way that the interactions of options can be taken into account. Both batch elimination and iterative elimination can be mapped easily to ATHENa and need only minor changes to handle options that are not binary as they were originally intended.

3. ATHENA

The open-source benchmarking environment called ATHENa (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. ATHENa aims to alleviate the complexity of this task, overcome the pitfalls and difficulties inherent to these evaluations, and to spread awareness of good practices for the benchmarking of algorithms belonging to the same class. The goal of this section is to describe the ATHENa environment and introduce a number of improvements to this system that can greatly increase its usefulness as a benchmarking tool.

What is ATHENa

ATHENa is at its most basic a set of Perl scripts that allow a user to easily run FPGA synthesis and implementation tools in the batch mode from the command line. It is free and open source software under development by the Cryptographic Engineering Research Group at GeorgeMasonUniversity. The scripts support tools from the two largest FPGA vendors, Xilinx and Altera and can utilize both licensed and unlicensed versions of the tools. Both VHDL and Verilog description languages are supported. ATHENa also supports automated functional verification of the designs and detailed report generation.

Design and Workflow

ATHENa allows a user to take VHDL codes that they have written and easily execute synthesis and implementation tools on these codes using configurations that have been developed by the ATHENa team as well as other users who have uploaded results to the ATHENa database. A designer can specify a number of options such as the devices they wish to target, timing constraints, generic values they wish to set and then execute the design tools for their device target through the command line and automatically generate results that can be uploaded to the ATHENa results database.

Strategies and Searches

One of the goals of the ATHENa environment is to provide a tool that aids in the optimization of design targeting FPGAs. These optimizations are carried out through the use of strategies and searches.

Strategies are predetermined option sets that target a certain performance metric such as throughput to area. The options have been determined to yield good results for the specific metric that they target making it easy for a designer to optimize their design for a specific metric even if they are ignorant to the underlying tools and how they work. While strategies do not guarantee best performance or optimal results for the metric that they target, they provide good targeted results with no effort or tool knowledge from the designer.

Searches are like strategies except that they aim to achieve more optimal results than strategies aim at. Instead of a static predetermined set of options the options and their values are programmatically determined. This process requires running the design tools multiple times varying the input options in response to the output performance results. Placement Search permits the exploration of result dependencieson the starting point of placement. This starting point is determined by the options of the FPGA implementation tools called: Cost Table in Xilinx tools and Seed in Altera tools. Cost Table can take any integer value between 1 and100, and Seed any value between 1 and 2^32. Placement search is done by exhaustively searching a subset of the valid placement values. Exhaustive search is an extension of placement search and includes options other than placement options. Any tool option can be evaluated in the exhaustive search, but the number must be constrained to a relatively low value because if not the search is unfeasible. The last search provided by ATHENa is frequency search. Both Xilinx and Altera tools have an option corresponding to the desired frequency of the resulting implementation. By adjusting this value the performance of the design can change leading to greater performance. Frequency search aims to determine the best value for input frequency that yields the best resultant performance.

Areas for Improvement

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. In the general case ATHENa executes all jobs in a serial batch manner. Limited parallelism existed with some search algorithms but is limited to the logical cores of the host system. It was determined that a way to extract the parallelism that exists in these jobs and distribute it across multiple compute nodes would greatly increase the effectiveness of the system. Many of the benchmarking tasks executed by ATHENa have few dependencies on other tasks and are candidates for parallel execution. By utilizing the parallelism offered by a distributed batch system this inherent parallelism can be exploited.

It was also determined that besides the nearly serial nature of the execution of tasks in ATHENa, more effective optimization algorithms should be developed that can make better use of the distributed system architecture and find a more efficient way to reach an optimized design solution. This shortcoming lead to the design of the optimization space exploration algorithms presented in this text as well as the Frequency Search and Placement Search that was designed with the distributed system in mind.

The last issue that needed to be addressed was the user interface of the system. ATHENa up to this point was a command line tool. The ability to effectively monitor and manage large benchmarking workloads was missing. In the design of the new ATHENa a GUI front-end was developed to help alleviate these issues and allow easy monitoring and management of ATHENa tasks.

By combining these new features ATHENa becomes much more powerful and practical to use. It also helps the system to become much more approachable from a user perspective and not just a pure engineering tool that requires knowledge of the implementation details to use effectively.

4. BATCH SYSTEMS AND HIGH THROUGHPUT COMPUTING

Benchmarking algorithms designed with hardware description languages and targeted to FPGA devices require the use of synthesis and implementation tools that can utilize a large amount of CPU time to complete. This fact is compounded by the need to run many benchmarks with separate option sets to comprehensively benchmark any single implementation. The result is that it can take many days and even weeks to properly synthesize and implement a candidate algorithm for evaluation across all of the option sets required. One solution to this problem is to utilize High Throughput Computing (HTC) architectures to help alleviate the massive CPU hours required to complete benchmarking and evaluation tasks.

HTC is the use of computing systems to maximize the number of operations that can be completed per month or year as opposed to how to minimize the latency of an operation. They are specifically computing environments that can deliver large amounts of processing capacity over long periods of time. They are specially suited for problems with long compute times that can be parallelized over the input data sets that need to be processed. Many systems exist for setting up HTC environments, but a subset, Batch Systems, are of interest to this study. Batch systems are systems that can distribute the execution of batch programs across multiple computing nodes simultaneously. They are primarily used in research environments where the same program needs to process a large set of varying data. This applies directly to the ATHENa problem space where the program is ATHENa and the data are the various option sets used to benchmark the cryptographic algorithms. By leveraging the parallelism offered by a HTC batch system, multiple benchmarking jobs can be run at once alleviating the time to comprehensively benchmark any single algorithm.

Batch System Comparison

In the case of ATHENa a HTC environment needs to be able to distribute and manage multiple benchmarking runs across heterogeneous computing resources. This will allow ATHENa to decrease the time needed to complete benchmarking substantially and increase the utilization of the existing resources. A set of requirements was gathered and ranked in importance to help compare various systems. Many different batch systems as well as other HTC environments exist and are widely used. Below is an investigation of a subset of such systems that were identified for their potential use with ATHENa.

Requirements

The requirements are listed below and given a weight that ranks their importance. Each system is given a score for each requirement and then the weighted result is used for final comparison. The weighting scheme used to determine the best system and the scores for the different systems that were evaluated are located in Table 1.

Architecture

The architecture of the system is important for ATHENa. It is desirable to have a system that has a centralized server that manages the batch jobs with compute and submit nodes that communicate with this server. The systems should impose no language restrictions on the jobs that are run and would preferably not require any special build steps for the ATHENa software. The last architectural requirement would be the ability of the system to transparently fall back to a single machine system if the user is not connected to the central server requiring only one ATHENa environment that would work in a HTC and single node system. This flexibility is paramount to making the system intuitive and easy to use.

Administration

The system should be easy to administer and setup as well as easy to use. It should also support remote administration allowing a user to schedule and monitor submitted jobs remotely. Once jobs are submitted a user of the system should be able to leave it unattended for long periods of time and monitor the submitted jobs from any machine with network access to the system. A graphical user interface to monitor and administer jobs would also be recommended.

Resource Specification

The batch system should be flexible enough to handle multiple types of resources. In the context of ATHENa, the resources would be available machines, processing cores and software licenses that are needed by the ATHENa programs.

Fault Tolerance

The system should be able to recover from basic errors and reschedule work accordingly. The fault tolerance needs to exist at both the central server if one exists and at the compute nodes that do the actual processing. In the most basic case the system needs to allow work to not be lost if the central server goes down, and the central server should be able to allow for nodes that execute jobs to leave the system without failure.

Security

The system should try to minimize the potential security threats inherent to distributed systems. Basic authentication and other measures must be present.

Stage In/Out

The system should allow files to be specified that must be transferred to execution nodes in order to process the batch job and files that must be transferred back upon completion. The system optionally should provide a mechanism to transfer the executable batch program with the input for maximum flexibility. The transfers would preferably use ssh/scp for transmission.

Platform Support

The batch system should support multiple operating systems. ATHENa currently runs on both Windows and Linux and the batch system should utilize both if possible.

Scheduling

Scheduling is a very important aspect of batch systems. Scheduling effects how the compute resources are used and to what extent the resources are utilized. The system should provide flexibility to define multiple scheduling options, change job priorities, pause/resume/terminate jobs, and define different user profiles.

Batch Systems Investigated

For the purpose of this investigation, six different batch systems where found and compared for use with ATHENa. While some of the systems are not pure batch systems all can be used for this purpose. All of the systems are FOSS (free open source software) systems and are under active development.

Condor

Condor is a HTC project under development by the University of Wisconsin-Madison department of computer science. The system was first put into production 15 years ago and is still under active development. Condor is a specialized workload management system for compute intensive jobs that provides job queuing mechanisms, scheduling policies, priority schemes, resource monitoring, and resource management. Condor can process serial or parallel jobs and can utilize idle machines for processing. Condor contains many other features as well as third party additions for things such as graphical displays and accounting. Stage In/Out is supported as well as the ability to transfer the application to be executed to the remote compute node. Condor also has the ability to run on a single machine in the absence of a cluster, though this behavior is not transparent to the user and requires specialized configuration. Condor is a multi-platform system and can run on many flavors of Linux, Solaris as well as Windows.

Globus

Globus is a tool kit for grid computing. Unlike the traditional batch systems Globus is a meta-scheduler, used to schedule work across multiple compute clusters. Globus can be adapted and used to administer a single compute cluster although with slightly less functionality than the full system. Globus provides a basic scheduler but does not provide a mechanism for data transfer without the existence of a shared file system. Third party tools have also been created for Globus for the monitoring and accounting of jobs.

Torque

Torque is a resource manager providing control over batch jobs on distributed compute nodes. Torque is a community effort that was based on the PBS (portable batch system) project. Torque has a basic scheduler but is primarily used as only a resource manager with an external scheduler. Torque provides job management, resource management, resource and job monitoring, job queuing and basic batch scheduling. Torque has stage in/out capabilities utilizing SCP or networked file systems. Torque can be configured on a standalone machine although is not intended to be configured this way.

Toque with MAUI Scheduler

The MAUI scheduler is an open source scheduler for clusters and super computers. It has an array of scheduling policies, can handle dynamic priorities, resource reservations, and other advanced scheduling capabilities. It also contains useful monitoring and diagnostic tools and keeps track of resource utilization. MAUI while not a batch system can be integrated into Torque, increasing Torque’s scheduling capabilities.

JPPF

JPPF is the Java Parallel Processing Framework. JPPF enables applications with large processing requirements to be run on any number of computers. JPPF is written in Java and requires jobs to wrapped in a simple java class for execution. Because JPPF is a Java framework any platform that supports Java can run JPPF jobs. There are extensive examples and third party tools for JPPF. While JPPF is a parallel processing framework it can easily be adapted for batch processing. While the documentation appears to be good, information was scarce on support for stage in/out and what is used for security and inter-node communications. Basic scheduling is supported with job prioritization and tools for monitoring and managing resources and job execution. JPPF also contained a full featured GUI for job management and monitoring, a feature that no other system had to this extent.

SLURM

SLURM (simple Linux utility for resource management) is an open source cluster management and job scheduling system for Linux clusters. It was developed by Lawrence Livermore National Laboratory and has been used on many projects. It provides exclusive or non-exclusive access of compute resources, a framework for starting, executing and monitoring work on the compute nodes, and it arbitrates

Table 1 Batch System Comparison

		Condor	Globus	Torque	MAUI	JPPF	SLURM
Architecture	Centralized Server(1)	5	5	5	5	5	5
	No Language Restrictions(2)	5	5	5	5	2	5
	No Special Build Steps(2)	5	5	5	5	3	5
	Transparent Single Machine Use(1)	3	0	0	0	4	0
Total	(30)	28	25	25	25	19	25
Administration	Ease of Use(1)	4	2	4	4	2	4
	Remote Admin(2)	4	3	4	4	4	4
	Remote Monitoring(2)	4	3	4	4	4	4
	GUI(1)	2	0	0	0	4	2
Total	(30)	22	14	20	20	22	22
Resource Specificati on	Machines(2)	5	0	5	5	3	5
	CPU Cores(1)	4	0	5	5	0	5
	Licenses(2)	5	0	3	4	0	3
Total	(25)	24	0	21	23	6	21
Fault Toleran ce	Server(1)	5	0	4	4	0	5
	Compute Node(1)	5	0	4	4	0	5
Total	(10)	10	0	8	8	0	10
Securi ty	System Security(2)	4	4	2	2	0	4
Total	(10)	8	8	4	4	0	8
Stage In/Out	Stage In/Out(2)	5	5	4	4	3	0
	Binary/Program Transfer(1)	5	0	3	3	0	0
	SSH/SCP(2)	0	2	5	5	0	0
Total	(25)	15	14	21	21	6	0
Platfor ms	Windows(2)	5	0	0	0	5	0
	Linux(2)	5	5	5	5	5	5
Total	(20)	20	10	10	10	20	10
Scheduling	Job Priorities(2)	5	0	5	5	5	5
	Flexibility(1)	4	0	2	5	2	3
	Manual Pause/Resume/Terminate(2)	5	0	5	5	5	5
	User Profiles(1)	5	0	0	5	0	3
Total	(30)	29	0	22	30	22	26
Overall Score	(175)	156	71	131	141	95	122

contention for resources by managing a queue of pending work. SLURM is intended as a resource management system with other utilities used for stage in/out and complicated scheduling.

5. ATHENA CONDOR ARCHITECTURE

ATHENa in the initial version is a non-distributed multiprocessing program that consists of a set of Perl scripts that are used to interact with FPGA synthesis and implementation tool command line interfaces. The scripts take user configurations provided in structured configuration files that specify tool options, project options and formulas for latency calculations that are used to manage the execution of the command line tools and provide detailed reports after execution has completed. The parallelism was limited to the number of processors on the host system and was thus limited in the ability to efficiently process large sets of configurations as needed in benchmarking tasks. Due to the computational load of running benchmarks with the ATHENa system, a new approach was adopted to deal with these shortcomings. Distributed batch systems would allow the existing ATHENa scripts to remain for the most part intact and leverage the parallelism provided by a compute cluster to scale the system to the extent that compute resources are available. Parallelism is extracted at the run level, each run is a job that can be submitted to a remote compute node for processing. This is extremely beneficial with respect to advanced benchmarks such as exhaustive search and other orchestration algorithms for optimization because each run or option set being investigated is an independent work unit that can leverage the parallelism provided by the new architecture.

It was found in the investigation of distributed batch systems that Condor was the candidate system that most completely met the various requirements that ATHENa needed. The adoption of a distributed architecture required an evolutionary revision in the design of the ATHENa system as detailed here.

Condor Architecture

Condor as described previously is a distributed batch system for high throughput computing under development by the University of Wisconsin Department of Computer Science[5],[6]. The system allows for the distribution of computing across a cluster of heterogeneous computing resources.

Class Ad Mechanism

Condor utilizes a specialized class ad mechanism, similar to newspaper classified ads, to match jobs to compute resources[7]. Class Ad Matchmaking as described in Matchmaking: Distributed Resource Management for High Throughput Computing allows a program to specify what requirements it has and properly allocate the processing to a compute node that contains these requirements. It is particularly effective in an environment where various dissimilar resources such as compute nodes and software licenses change between available and unavailable states without advance notice. This is particularly important with respect to ATHENa because it needs to be deployed in an environment where compute nodes are not necessarily dedicated to processing

ATHENAjobs and the availability of software licenses needed for the design tools being executed can change continually.

Condor Pool

Condor organizes the processes and compute resources at its disposal into what is referred to as a condor pool. A condor pool consists of a single machine known as the central server and an arbitrary number of machines that have joined the pool as either a submit or execute host. Conceptually the pool is a collection of resources and resource requests. The role of Condor is to match the requests to the appropriate resources for processing. All of the machines that are part of the Condor pool send periodic updates to the central server which is the centralized repository for all pool state information.

Each resource, or execute host within the Condor pool has an owner that defines the policies by which that particular resource will be used. These policies can include things such as the number of slots available to Condor and the software and licenses provided by the resource. Slots are analogous to processors and represent the number of jobs a resource will execute simultaneously. Condor can be configured to minimize the impact to an owner of a resource while in use or leverage as much processing power as possible. The policies and specific attributes of the resource are defined in Condor ClassAds.

Resource requests, or jobs are the units of work that are distributed to the execute hosts for processing. Each resource request has an owner, the individual who submitted the resource for execution. Resource requests are defined in Condor submit files that represent the requirements that the resource request has. These requirements can be very simple or complex. A description of the Condor submit files and an example of one generated by ATHENa are in the appendix.

Condor Master

The condor_master daemon is a software process that is responsible for maintaining the availability of the various software processes needed by a Condor pool running on all of the nodes within a condor pool.

Condor Startd

The condor_startd daemon is the process responsible for representing a computing resource to the Condor pool. It advertises the attributes and capabilities of a resource through the ClassAds defined by the owner of the resource. It is responsible for providing the information to the central manager that it needs to match resource requests to a particular resource. Condor_startd will run on all machines in the condor pool that wish to be an execute host. When a resource request has been matched to the resource represented by condor_startd it spawns the condor_starter process as described below.

Condor Starter

The condor_starter process is the process that spawns the job that is to be executed remotely on an execute host. It sets up the environment needed by the job and monitors the job while it is running. It is also responsible for sending status about a job back to the submit host that submitted the job when the job completes.

Condor Schedd

The condor_schedd daemon is the process responsible for representing resource requests to the Condor pool. The condor_schedd daemon needs to be running on any machine that wishes to play the role of a submit host in the Condor pool. The condor_schedd stores all jobs submitted by users in a job queue that is managed by the schedd. The condor_schedd advertises the jobs in the job queue to the Condor pool and is responsible for claiming the resources needed to serve the job requests. Once the condor_schedd has been matched with a particular resource it spawns the condor_shadow process as described below to serve the request.

Condor Shadow

The condor_shadow process is responsible for managing a remote job that is executing on a remote host. It handles file input and output for the remote job and handles retrieving the results produced by a remote job and delivering them to the submit host that submitted the job.

Condor Collector

The condor_collector process runs on the central server of a Condor pool and collects all of the information about the state of the Condor pool. It is responsible for managing all of the ClassAd updates from the resources that comprise the pool. It also maintains the information needed for all submit hosts to communicate with all of the execute hosts in the pool.

Condor Negotiator

The condor_negotiator is the intelligence that manages resources and resource requests within the pool. It is responsible for all matchmaking in the pool. The process that the negotiator uses to match resources and resource requests is known as a negotiation cycle. Periodically the negotiator begins a negotiation cycle where it queries the collector for the state of the pool. It then contacts each condor_schedd that has jobs in its job queue and tries to match the pending jobs with available resources in the pool. The negotiator enforces the priorities of all users and jobs within the pool and users this information as a basis for scheduling. The negotiator runs on the central manager.

Roles

There are three roles that a machine can take on in a Condor pool with respect to ATHENa. These roles define what daemons and processes run on the machine and what capabilities the machine has. In addition to being able to take on a single role in the pool a machine can encompass multiple roles at once. A machine can be both a submit host and an execute host. A single machine pool can also be configured where a single machine takes on all roles and runs in isolation.

Central Manager

The central manager is the machine that contains the scheduling ability within the pool. It collects all of the state information about the resources provided by execute hosts and the pending jobs offered by the submit hosts. It's job is to effectively match resource requests with the available resources while enforcing the policies provided by the ClassAds in the system. The processes that need to run on the central manager include the condor_negotiator, the condor_collector, and the condor_master. A Condor pool generally contains a single central manager although it can contain more for redundancy in the case that the central manger becomes inoperable.

Submit Host

Submit hosts are machines that users submit jobs from. The submit hosts run the condor_schedd daemon and host the condor_shadow process for each remote job that has been submitted by that particular submit host. The submit host communicates pending jobs and their requirements to the Condor pool for consumption by the central server for matchmaking. It is also responsible for file I/O to the remote jobs that it manages.

Execute Host

Execute hosts are machines in the Condor pool that actually process the resource requests made by submit hosts. The condor_startd daemon and condor_starter run on the execute hosts as well as the remote job specified by a resource request. The execute host communicates state and resource information to the central manager via Condor ClassAds. The execute host manages the jobs that are matched to it and provide job status to the central manager as well as the submit host that submitted the job. Figure 1 shows the different roles within a condor pool and how they interact. The daemons that make up the roles are listed for each.

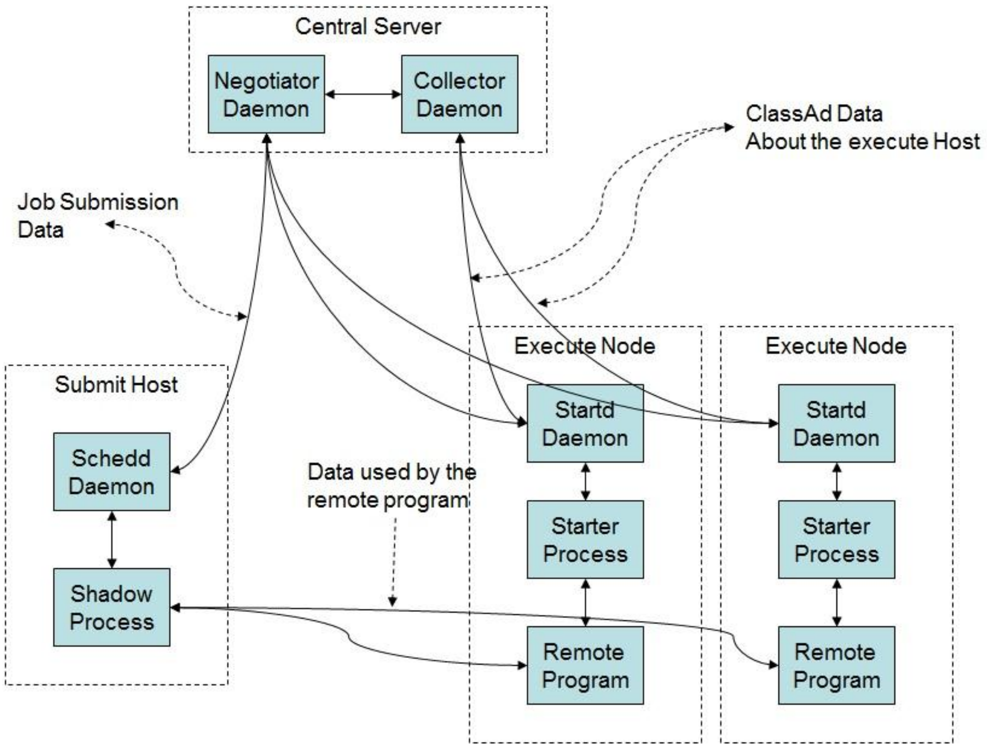
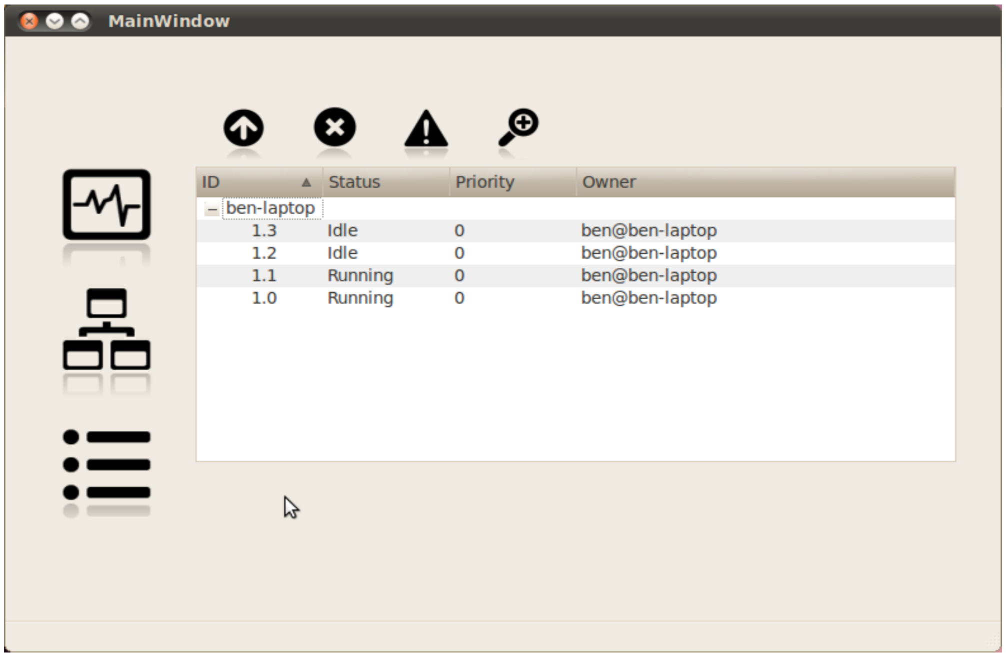


Figure 1 Condor Roles

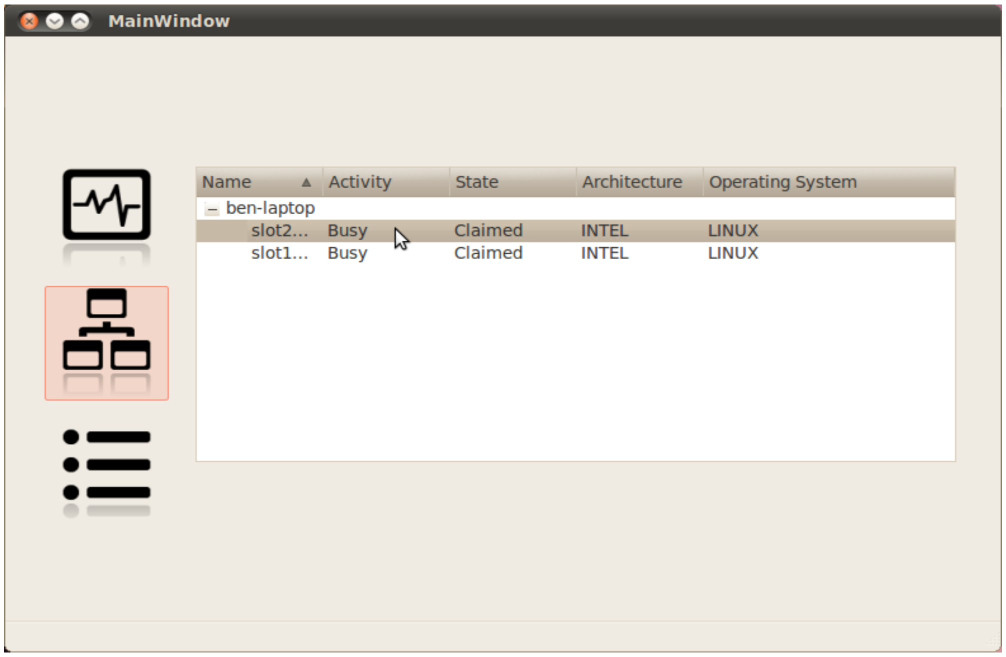
ATHENa Client/Server Architecture

Enhanced ATHENa was designed from the beginning to utilize parallelism as much as possible. The enhanced ATHENa software is developed in Python and follows object oriented design principles. The new design was divided into two distinct pieces, a client application where a user submitted benchmarking jobs, and a server side application that processes the jobs. The client application consists of a GUI for submitting and monitoring jobs. Information regarding each job is displayed as well as controls to pause, resume, and cancel existing jobs. Behind the GUI is an application that extracts the relevant information from the job configurations then sends the job to the server side application via the Condor batch system. The application will determine if a user submitted job can be broken up into independent subtasks for processing in parallel and send these subtasks to the batch system separately. The client application resides on a machine that takes on the submit host role within the system. The application generates Condor submit files for scheduling and interacts with the Condor daemons to schedule, process, and monitor the jobs submitted by a user. The new design space exploration algorithms that are explained in more detail to follow are also a part of the client application. By residing on the client, the algorithms can utilize the parallelism offered by Condor. Figure 2 and Figure 3 are screenshots of the GUI designed for the client application. Figure 2 shows the job monitoring table. The table lists what jobs have been submitted, by whom, and what state they are currently in. Figure 3 is the node monitoring table. It displays data on the connected execute hosts in the condor pool. Information is displayed about the operating system, CPU architecture as well as the state of the execute hosts.



ID	Status	Priority	Owner
ben-laptop			
1.3	Idle	0	ben@ben-laptop
1.2	Idle	0	ben@ben-laptop
1.1	Running	0	ben@ben-laptop
1.0	Running	0	ben@ben-laptop

Figure 2 Client GUI Submissions



Name	Activity	State	Architecture	Operating System
ben-laptop				
slot2...	Busy	Claimed	INTEL	LINUX
slot1...	Busy	Claimed	INTEL	LINUX

Figure 3 Client GUI Execute Hosts

The server side application 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. The server side application is executed by the batch system when a job is submitted and matched to the compute node that the server side application resides on. The server application resides on a machine that takes on the Execute Host role in the system. Results are generated by the application and the batch system is notified of job completion and collects these results for transporting back to the machine that the user submitted the job from. Figure 4 shows the general dataflow and architecture of the combined ATHENa Condor Client/Server system. The User Job is the configuration file used as input by the user. The Submission generator extracts subtasks from thee User Job. These subtasks are described in the Condor Submission that is consumed by the Central Manager and used for matching the jobs to an appropriate execute host. The execute hosts communicate classAds as well as state information to the Central Manager to aid in the Match Making process. Once the jobs have been matched to execute hosts, the ATHENa server application executes the design tools with the proper command line options to generate results. Once the results are generated the server application packages the data for consumption by the client application for report generation.

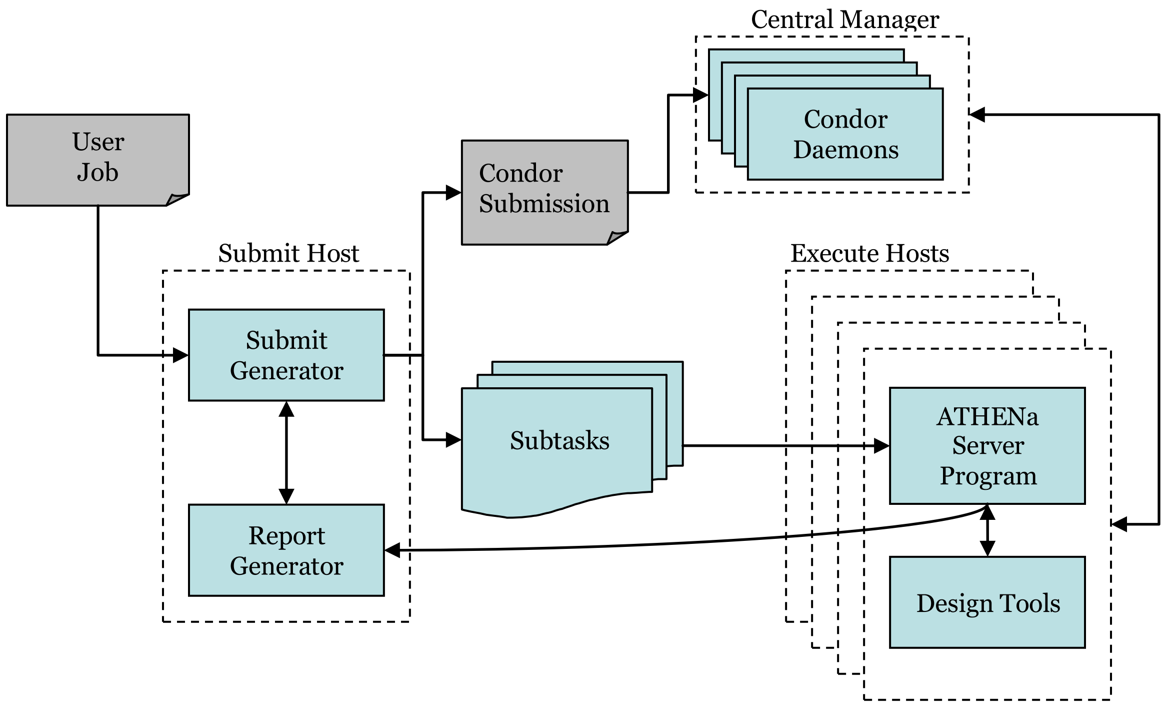


Figure 4 Client/Server Architecture

6. RELATED TOOLS

There are other systems that aim to accomplish similar goals to the ones being accomplished with enhanced ATHENa. The main advantages of ATHENa over most other offerings 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.

eBACS

eBACS (ECRYPT Benchmarking of Cryptographic Systems)[2], [3] 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. eBACS contains the eBATS(ECRYPT Benchmarking of Asymmetric Systems), eBASC(ECRYPT Benchmarking of Stream Ciphers), and eBASH(ECRYPT Benchmarking of All Submitted Hashes) systems. The eBASH portion of the system compares hashes, including the SHA-3 candidates utilized in the experiments for ATHENa. eBASH has already collected 51 implementations of 28 different hashes from 14 different families. These have been benchmarked on 69 different computers recompiled with 1201 different options. The system defines a standardized Application Programming Interface (API) for the supported cryptographic primitives. The goal of the system is to determine the best option set for each design on each machine architecture investigated. The concept and goal for ATHENa was inspired by the eBACS system, to remove the burden of generating high quality benchmarking results from the algorithm designer and generate benchmarking results in a clear unbiased manner for publication and comparison. The biggest difference between the systems is the software focus of eBACS. The job of optimizing the software algorithms is also simpler for eBACs because of their focus on one compiler, gcc. Software benchmarking and optimization also has an advantage in that one tool is used for optimization, the compiler, where with FPGA optimization there are multiple tools that need to be optimized to reach a final solution, Synthesis, Mapping, Placement, and more depending on the specific tool chain used.

Xilinx PlanAhead

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 can be deployed in a distributed cluster environment to increase the throughput of the benchmarking tasks. PlanAhead also contains predefined option sets for benchmarking purposes. One of the main differences between the optimization techniques employed by PlanAhead and those used by ATHENa is the isolation of each specific tool in PlanAhead. PlanAhead optimizes each stage of the tool chain in isolation, reducing the total number of runs to explore the same search space. Enhanced ATHENa has limited support for this capability, allowing each tool to be run in isolation but providing little optimization algorithm support other than Most Effort optimization while in this mode. PlanAhead while similar to ATHENa does not support any other vendor device or tool chains limiting its use to Xilinx devices and tools alone.

Altera Design Space Explorer

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. Different optimization targets are predefined that choose option sets to achieve performance relative to a given metric. Design Space Explorer can optimize for area, speed, and power. Design Space Explorer can also work on multiprocessor systems as well as distributed cluster environments. As with the Xilinx specific tool PlanAhead, Design Space Explorer only utilizes Altera devices and the Altera tool set to optimize designs.

7. OPTIMIZATION SPACE EXPLORATION

Least Effort

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. Running tests with this method will help establish a baseline for the minimum time that you can spend tuning and what sort of results you could expect. It serves as a useful baseline for comparison of all other optimization methods. The exact options and the states that are used are described in the case studies.

Most Effort

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. While limited in use it is incredibly powerful for a small search space and will be used to compare the effectiveness of the other algorithms using a constrained set of options.

Batch Elimination

The Batch Elimination algorithm proposed here is adapted from previous research in compiler option selection [8]. 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. It will be shown through experimentation that a near optimal solution can be achieved if there is minimum interaction between options for the tested design.

The algorithm itself is quite simple in design. The effect of one option, O_i is represented by a metric called Relative Improvement Percentage, $RIP(O_i)$, 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.

Equation 1 Relative Improvement

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

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 as:

Equation 2 Relative Improvement Baseline

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

If, $RIP(O_i = 1) > 0$, the option O_i has a positive effect. The Batch elimination algorithm enables this option in the final option set. The pseudo code 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 $RIP(O_i = 1)$.
3. Enable all options with positive RIPs. 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. To help illustrate the example Table 2 represents the option set at each step of the batch elimination algorithm. 0 represents the default option where a 1 represents a non-default option state. A + for the Relative improvement denotes an improvement over the baseline and a – denotes none, or negative impact.

Table 2 Batch Elimination

Run	Option 1	Option 2	Option 3	Option 4	Option 5	Relative Improvement
Ob	0	0	0	0	0	N/A
O1	1	0	0	0	0	+
O2	0	1	0	0	0	-
O3	0	0	1	0	0	-
O4	0	0	0	1	0	+
O5	0	0	0	0	1	+
Of	1	0	0	1	1	N/A

Note: 1 Ob is the baseline, Of is the final run

Batch Elimination takes *n+1* runs of the design tools to complete where *n* is the number of options being investigated but can be completed in the equivalent of 2 runs by utilizing *n* compute resources. Figure 5 illustrates the task decomposition of the Batch Elimination algorithm.*Ob* is the baseline and *O1-O5* are the runs representing the optios being investigated.

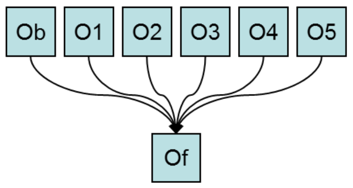


Figure 5 Batch Elimination Parallel Task Decomposition

Iterative Elimination

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. Iterative elimination requires at most $n * \frac{n}{2} + \frac{n}{2}$runs to complete while on average it can be much less. The following pseudo code 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 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 $RIP(O_i = 1)$.
3. Find the option O_i , that has the greatest RIP. Create a new baseline with this option set to on instead of off.
4. Repeat the previous steps until there is no positive RIP for any option.

The final baseline will represent the fully optimized option set. To help illustrate the example Table 3 represents the option set at each step of the iterative elimination algorithm. 0 represents the default option where a 1 represents a non-default option state.

A + for the Relative improvement denotes an improvement over the baseline and a – denotes none, or negative impact.

Table 3 Iterative Elimination

Run	Option 1	Option 2	Option 3	Option 4	Option 5	Relative Improvement
Ob	0	0	0	0	0	N/A
O1	1	0	0	0	0	+5
O2	0	1	0	0	0	-
O3	0	0	1	0	0	-
O4	0	0	0	1	0	+7
O5	0	0	0	0	1	+2
Ob2	0	0	0	1	0	+7 from baseline 1
O1	1	0	0	1	0	+3
02	0	1	0	1	0	-
03	0	0	1	1	0	-
04	0	0	0	1	1	+1
0b3	1	0	0	1	0	+3 from baseline 2
O1	1	1	0	1	0	-
O2	1	0	1	1	0	-
O3	1	0	0	1	1	-
Of	1	0	0	1	0	+10 total

Iterative Elimination can take as little as $n + n - 1$ runs of the design tools to complete and up to $n * \frac{n}{2} + \frac{n}{2}$ where *n* is the number of options being investigated. The algorithm can be realized in at most *n* runs if there are at least n compute resources available. Figure 6 illustrates the task decomposition of Iterative elimination. The ellipses is used to signify the unknown number of runs required to complete the optimization. *Ob-*

Obn represent the baselines used at each step of the Iterative Elimination process. *Of* represents the final optimized option state.

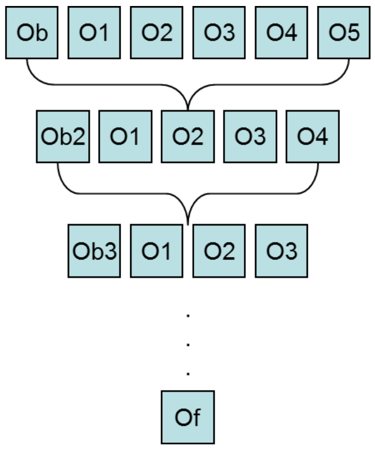


Figure 6 Iterative Elimination Parallel Task Decomposition

Orthogonal Array

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 [9], [10]. In our experiment we have n factors being considered creating an optimization space with 2^nsettings. 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 *N x k* matrix where the columns represent optimization options and the rows represent the settings used for each experiment. The matrix is filled with 1’s and 0’s 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 with eight rows corresponding to eight separate experiments, or sets of options. The array contains five columns, representing a search space of five options. Figure 7 shows the orthogonal array used in the Orthogonal Array Optimization implementation demonstrated in this thesis.

1	0	0	0	0
0	1	0	1	0
1	1	1	0	1
0	0	1	1	1
1	0	0	0	1
0	1	0	1	1
1	1	1	0	0
0	0	1	1	0

Figure 7 Orthogonal Array

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 optionO_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.

Equation 3Relative Improvement OA

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

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 tradeoff 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.

To help illustrate the example Table 4 represents the option set at each step of the orthogonal array algorithm. 0 represents the default option where a 1 represents a non-default option state. The Of option state is the final option state determined by the calculated RIP of the previous runs.

Table 4 Orthogonal Array

Run	Option 1	Option 2	Option 3	Option 4	Option 5
O1	1	0	0	0	0
O2	0	1	0	1	0
O3	1	1	1	0	1
O4	0	0	1	1	1
O5	1	0	0	0	1
O6	0	1	0	1	1
O7	1	1	1	0	0
O8	0	0	1	1	0
Of	1	1	0	0	1

Orthogonal Array optimization using the previously illustrated orthogonal array takes 9 runs of the design tools to complete. By running jobs in parallel with the distributed batch system, the algorithm takes the equivalent of 2 tool executions to complete running up to 8 instances in parallel if there are available compute resources. Figure 8 is the task decomposition of the Orthogonal Array optimization. *O1-O8* represents the options combinations used to determine the *RIP* of each option. *Of* represents the final option state that results from the optimization.

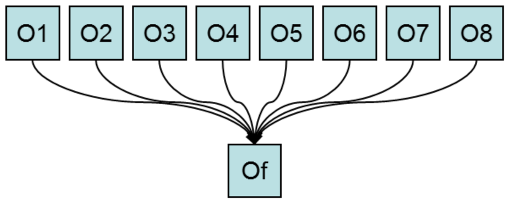


Figure 8 Orthogonal Array Parallel Task Decomposition

Frequency Search

Frequency search 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. In all of our experiments it is the default option set. All options 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.$Fin_n = Fout_0 \times [1 + (.1 \ \times n)]$for *n from 1 to 10*
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. $Fin_n = Fin \times [1 + (.03 \ \times n)]$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. $Fin_n = Fin \times [1 + (.01 \ \times n)]$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.

Frequency Search requires 21 full runs of the design tools to complete. Utilizing the distributed batch system frequency search takes the equivalent to the time to execute the design tools 4 times if at least 10 compute resources are available.

Placement Search

Placement search 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 the value for the placement option is used to determine the next set of values to be searched. A second set of placement values is searched centered on the best value experimentally determined from the first set. The new values are *+/- 5, +/- 4,+/- 3, +/- 2, +/- 1* from this value. If the best value from the initial set was 100, then only -5,-4,-3,-2,-1 are used. If the best value from the initial set was one, then +1,+2,+3,+4,+5 are used. As an illustration if the value for placement experimentally determined in the first set was 50 then the second set would be 45, 46, 47, 48, 49, 51, 52, 53, 54, 55. After running the second set, the value for placement that lead to the greatest resultant performance is the result of the search. The Placement Search algorithm can complete in the time it takes to run two sequential runs if there are at least 11 compute resources.

Tool Separation

Tool separation is a method that isolates each tool and performs an optimization per tool as opposed to for all tools at once. This method can limit the search space that is tested by each algorithm by limiting the search space to only those for a specific tool. While in theory this is a good way to limit the search space, it has drawbacks as well. With respect to the Xilinx tool chain problems arise with the output generated at each stage. The Xilinx synthesis tool generates a synthesis frequency, but in most cases it is not apparent or consistent how the generated synthesis frequency affects the outcome of other stages that follow. The highest synthesis frequency does not in all cases lead to the highest implementation frequency. The map tool will not generate timing information unless a certain option is used that is not compatible with many other map tool options.

Tool separation is also not an optimization method used but a technique to apply optimization algorithms. Currently only Most Effort optimization is supported due to the difficulty in determining best option sets at each stage. Adapting the other algorithms presented here, or developing new algorithms that take advantage of this technique could be an area of future work. The advantage of separation is the ability to isolate the optimization of each tool from each other, providing means to potentially search more options with fewer runs. This concept hinges on the idea that each successive stage is best optimized using the most optimal result from the previous stage. While this seems reasonable it is not always the case and can cause problems when optimizing in this manner.

Metrics

The metrics used by ATHENa for optimization are throughput, the ratio of throughout to area, and area. Area is defined by the number of slices used by a design. Each algorithm aims to optimize one of these metrics as a target. This is done by calculating the necessary metric from the results and using it as input to the following optimization stages. Even in the evaluation of frequency search where desired frequency is being manipulated the metric being used to determine the best input frequency could be area or throughout to area ratio instead of the more directly related throughput.

Throughput is used to illustrate the raw computational speed of a design. It is useful in determining which algorithm may be the fastest. Formulas entered in the configuration files are used to calculate throughput in the results.

Area is used to illustrate how much circuit real estate a design will cost, how much hardware resources are used to implement the design. It is useful to determine the smallest design in a comparison. Area can be calculated using a number of different metrics reported by the synthesis and implementation tools. LUTs or CLB Slices are commonly used while other metrics such as the number of DSPs or number of BRAMs used could be considered as well but have some issues when making comparisons. The use of BRAMs or DSPs when comparing area is difficult because it is hard to make a direct comparison between BRAMS or DSPs and slices/LUTs to determine which design is technically smaller.

The ratio of throughput to area is used to show which designs exhibit the best balance between raw speed and hardware cost. For the case studies included in this paper this ratio is used for comparison.

8. SHA-3 CASE STUDY

SHA-3 Candidates

The SHA-3 competition is in the final round of evaluation in determining the next Cryptographic Hash Algorithm to be adopted as a NIST standard. This final round contains a total of five algorithms, four of which are evaluated in the enhanced ATHENa experiments.

BLAKE is a cryptographic hash algorithm designed by Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. It is based on the ChaCha stream cipher developed by Daniel Bernstein and comes in two variations, 256 and 512 using 32-bit and 64-bit words respectively.

JH is a cryptographic hash algorithm submitted to the NIST SHA-3 competition by Hongjun Wu. The algorithm produces a digest of 224, 256, 384 or 512 bits.

Keccak is a hash algorithm designed by Guido Bertoni, Joan Daemen, MichaëlPeeters and Gilles Van Assche. The algorithm has been measured to be notably faster in ASIC implementations in hardware than the other candidates as shown in "Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations."Keccak makes use of the sponge construction and is hence a sponge function family.Keccak is a joint design by STMicroelectronics and NXP Semiconductors.

Skein was created by Bruce Schneier, Stefan Lucks, Niels Ferguson, Doug Whiting, MihirBellare, Tadayoshi Kohno, Jon Callas and Jesse Walker. Skein is based on the Threefishtweakable block cipher. Skein supports internal state sizes of 256, 512 and 1024 bits, and arbitrary output sizes.

Experimental Design

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 four proposed algorithms from the SHA-3 cryptographic hash algorithm competition, BLAKE, JH,Keccak, and Skein and targeted two Xilinx devices, a Spartan 3 device, xc3s1000fg676-5, and a Virtex 6 device, xc6vlx75tff784-3. The architectures chosen for evaluation were the basic iterative architectures as implemented by the Cryptographic Engineering Research Group at George Mason University [11], [12]. The basic iterative architectures were chosen because the designs were previously benchmarked by the ATHENa system showing good performance improvement through optimization. The two devices were chosen to show how benchmarking and optimization tasks change when pairing different devices and algorithms. The experiments were conducted using Xilinx ISE version 13.1. The enhanced ATHENa distributed environment was deployed on two eight-core workstations that provided sixteen compute resources for use.

In addition to the device and algorithms, different option sets were used to illustrate different properties of the system. The first experiment is meant as a test to determine the ability of the orchestration algorithms to achieve near exhaustive search performance in much fewer runs. Since an exhaustive search of the optimization space is being utilized, the search space necessarily needs to be small for this initial experiment. A small constrained option set including only 5 options was chosen for both the Spartan 3 and Virtex 6 experiments. An expanded option set of 9 options was chosen for each device for the second experiment 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.

Table 5 Experiment 1 Options

		XST			**Map**	**Par**
Spartan 3	Options	opt_level	opt_mode	cross_clock_analysis	ol	ol
	Least Effort	2	speed	no	high	high
	Algorithms	1, 2	Speed, area	No, yes	Std, high	Std, high
Virtex 6	Options	opt_level	opt_mode	lc	ol	ol
	Least Effort	2	speed	off	high	high
	Algorithms	1, 2	Speed, area	Off, area	Std, high	Std, high

Table 6 Experiment 2 Options

		XST				Map				Par
Spartan 3	Options	Opt_level	Opt_mode	cross_ clock_ analysis	netlist_ hierarchy	ol	pr	ogic_ opt	equivalent_ register_ removal	ol
	Least Effort	2	speed	no	rebuilt	high	b	on	on	high
	Algorithms	1, 2	Speed, area	No, yes	As_optimized, rebuilt	Std, high	b, off	On, off	On, off	Std, high
Virtex 6	Options	Opt_level	Opt_mode	lc	netlist_ hierarchy	ol	pr	ogic_ opt	global_opt	ol
	Least Effort	2	speed	off	rebuilt	high	b	on	speed	high
	Algorithms	1, 2	speed, area	off, area	as_optimized, rebuilt	std,high	b, off	on, off	speed,off	std, high

Results

The JH algorithm results are shown in detail in Table 7. For Experiment 1 targeting the Spartan 3 device Batch Elimination produces a 5.3% performance increase, 11% below the optimal increase achievable. Iterative Elimination produces a nearly optimal result only .3% below the optimal performance, showing the ability of the Iterative Elimination algorithm to produce better results than Batch Elimination. Iterative Elimination achieves this result while executing the design tools only 9 times, nearly one third the amount needed for Most Effort Optimization. The Orthogonal Array algorithm produces a near optimal 15.5% performance increase.

The results of Experiment 1 for JH targeting the Virtex 6 device show a much better improvement for Batch Elimination, 8.6%, only 4.9% below the optimal solution of 13.5%. Iterative Elimination reaches the optimal solution in 14 runs, less than half the

Most Effort optimization requires. The Orthogonal Array algorithm also produces an optimal result and does so in a minimal 9 runs.

Experiment 2 targeting the Spartan 3 device shows an improvement of 18.6% by just moving to the larger option space. Batch Elimination leads to an improvement of 11.4%, less than Least Effort with 9 options but 6.1% higher than the initial Batch Elimination with 5 options. Iterative Elimination improves the performance to 21.9% above the base line. The Frequency Search Algorithm provides the greatest performance improvement of 42.5%. Both Batch Elimination and Iterative Elimination fail to improve the overall performance after the Frequency Search leading to results of 25.7% and 38.4% respectively. Batch Elimination is improved to 35.4% with a final placement search while Iterative Elimination is unchanged. It should be noted that Frequency Search yields the greatest performance in terms of throughput to area ratio, the metric that is the focus of the optimization, however, applying the design space exploration algorithms to the frequency search results leads to greater throughput performance at the cost of area. The final optimized design outperforms the results generated by the old ATHENa environment by 15% and achieves this result in only 13 more sequential runs.

Targeting the Virtex 6 device, Experiment 2 shows identical results for both Batch Elimination and Iterative Elimination. This can be attributed to the optimization options common between the 5 and 9 option sets. Frequency Search yields a performance increase of 31.8% and subsequent applications of Batch Elimination and Iterative Elimination yield 27.4% and 47.8% respectfully. Iterative Elimination increases performance over that achieved by frequency search by 16% while Batch Elimination decreases this performance by 4%. Placement Search yields performance increases of 51.9% and 54.4% for each set. The final performance increase for JH targeting the Virtex 6 device is the one case where the enhanced ATHENa environment underperforms the old ATHENa environment by a significant margin, having performance 16% lower with a sequential runtime 8 runs longer.

Table 7 JH Results

	Algorithm	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs (parallel)	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs
		Spartan 3					Virtex 6				
Experiment 1	LE	4084	1530	0.375	0.0	1	1170	4370	3.735	0.0	1
	ME	3525	1543	0.438	16.8	32	1024	4342	4.241	13.5	32
	BE	3621	1428	0.394	5.3	7(2)	1080	4379	4.055	8.6	7 (2)
	IE	3525	1532	0.435	16.0	11(2)	1024	4342	4.241	13.5	15 (3)
	OA	3520	1524	0.433	15.5	9(2)	1024	4342	4.241	13.5	9(2)
Experiment 2	LE9	3471	1543	0.444	18.6	1	1487	4368	2.937	-21.3	1
	BE9	3471	1449	0.417	11.4	11(2)	1080	4379	4.055	8.6	11(2)
	IE9	3383	1545	0.457	21.9	40(5)	1024	4342	4.241	13.5	34(4)
	FS	3337	1782	0.534	42.5	21(4)	979	4820	5.845	31.8	21(4)
	FS=>BE9	3564	1680	0.471	25.7	32 (6)	906	4312	5.696	27.4	32(6)
	FS=>IE9	3602	1869	0.519	38.4	48(7)	911	5030	5.905	47.8	55(8)
	FS=>BE9=>PL	3612	1833	0.508	35.4	53(8)	884	5015	5.807	51.9	53(8)
	FS=>IE9=>PL	3602	1869	0.519	38.4	69(9)	911	5252	6.279	54.4	76(10)
	GMU-1	3913	1804	0.461	23.1	56	849	5412	6.375	70.7	68
	XPA	3690	1671	0.453	20.9	54	1024	4182	4.084	8.2	54

The experimental results for the BLAKE design are shown in detail in Table 8. The Batch Elimination algorithm generates results that are 7.9% better than the Least Effort optimization and 25.1% less than the optimal solution for Experiment 1 targeting the Spartan 3 device. The less than optimal result can be attributed to interaction effects within the option space for this design and device target. Interaction effects are side effects that occur when two options are used simultaneously. The Batch Elimination algorithm does not handle these types of option interactions. Iterative Elimination produced an optimal solution, matching the performance of the Most Effort algorithm. The Orthogonal Array algorithm produced a decrease in performance of around 3%.

For Experiment two the Batch Elimination and Iterative Elimination algorithms targeting the Spartan 3 device perform exactly as in Experiment 1, leading to the conclusion that the options added for Experiment 2 have no performance impact in this design device pairing. Frequency Search leads to a 15% performance increase over the Least Effort baseline results, showing the effect of input frequency on results. Both Batch Elimination and Iterative Elimination cause a degradation of results when applied following the frequency search. A final 18% and 17% performance increase are recorded for Batch Elimination and Iterative Elimination respectively after the final placement search is applied. The failing of the larger option space to yield an increase in performance over the Experiment 1 option space is due to complex interaction effects as described earlier. With more options, the likelihood of interaction effects impacting performance is higher. Iterative Elimination is built to handle interaction effects better than Batch Elimination and this is shown to be true in all cases.

The Batch Elimination algorithm performs closer to expectations for the BLAKE design targeting the Virtex 6 device. Batch Elimination results in a 26% performance increase for Experiment 1, only 10% below the optimal value achieved by Most Effort optimization. As in the Spartan 3 device instance Iterative Elimination results in an optimal solution with a performance increase of 36%. The Orthogonal Array algorithm performs slightly better than the Batch Elimination algorithm with a performance improvement of 26.5%.

Experiment 2 yields some of the interaction effects that were experienced targeting the Spartan 3 device. The expanded option set yields only 15% increase in performance for Batch Elimination, 11% below that achieved with the smaller option set. Iterative Elimination overcomes the interaction effects experienced by the Batch Elimination algorithm but reaches the exact same performance as the initial 5 option set. Frequency search increases the performance by 43% over the Least Effort baseline performance. Applying Batch Elimination after the frequency search causes the performance to drop down to 22% due to option interaction effects. Iterative Elimination leaves the performance unchanged. The final Placement Search increases the performance in both cases for a final fully optimized design that leads to a performance increase of 44% using Iterative Elimination and an increase of 24% using Batch Elimination techniques.

In comparison to the two other optimization methods, Xilinx PlanAhead and the old ATHENa using GMU_Optimization_1, the performance of the new algorithms is better. Targeting the Spartan 3 device the new algorithms provide a maximum performance advantage of nearly 30%. The number of runs used to determine this value is lower, requiring only a frequency search to achieve, resulting in 21 runs. The fully optimized design has a performance increase of 15% using 69 runs, slightly more than either of the two competing techniques. Targeting the Virtex 6 device the performance increase is about 1%, and it takes 93 runs to reach this performance, much more than the other two techniques. While the algorithms do use more runs when compared to the other tools in terms of sequential runs, the actual runtime utilizing the 16 compute node system was 9 and 13 runs respectively leading to an actual runtime significantly lower than the other algorithms.

Table 8 BLAKE Results

	Algorithm	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs (parallel)	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs
		Spartan 3					Virtex 6				
Experiment 1	LE	641	150	0.234	0.0	1	199	239	1.656	0.0	1
	ME	467	146	0.312	33.0	32	145	328	2.259	36.4	32
	BE	502	127	0.253	7.9	7 (2)	158	331	2.093	26.4	7
	IE	467	146	0.312	33.0	11 (2)	145	328	2.259	36.4	18
	OA	487	110	0.226	-3.04	9 (2)	155	324	2.096	26.5	9 (2)
Experiment 2	LE9	641	150	0.234	0.0	1	212	346	1.631	-1.5	1
	BE9	502	127	0.253	7.9	11 (2)	187	356	1.906	15.1	11 (2)
	IE9	467	146	0.312	33.0	19 (2)	145	328	2.259	36.4	34 (4)
	FS	611	165	0.270	15.5	21 (4)	205	486	2.373	43.3	21 (4)
	FS=>BE9	479	118	0.246	5.3	32 (6)	180	364	2.024	22.3	32 (6)
	FS=>IE9	619	163	0.263	12.7	48 (7)	205	486	2.373	43.3	70 (11)
	FS=>BE9=>PL	477	131	0.276	18.0	53 (8)	171	353	2.066	24.7	53 (8)
	FS=>IE9=>PL	602	164	0.273	17.1	69 (9)	203	484	2.389	44.3	91 (13)
	GMU-1	641	154	0.240	2.5	57	182	432	2.376	43.5	57
	XPA	599	137	0.228	-2.7	54	207	422	2.039	23.1	54

Skein targeting the Spartan 3 device for Experiment 1 shows a 3.2% and 5.9% increase in performance for Batch Elimination and Iterative Elimination Respectively. Both of these algorithms fall far short of the optimal result of 18.4%. Orthogonal Array results again show an overall decrease in performance of almost 2%.

Experiment 2 targeting the Spartan 3 device shows an improvement of 11.8% and 13.03% for Batch Elimination and Iterative Elimination respectively. These results still fall short of the optimal 5 option results but offer a 2% and 4% increase over the Least Effort Optimization using 9 options. Frequency Search leads to an increase in performance of 16.3%. The final optimization algorithms all achieve the same result and are unable to improve upon Frequency Search achieving an overall performance increase of 15.3%, 1% lower than achieved with Frequency Search alone. In comparison to the old ATHENa environment and increase in 4% is achieved.

Experiment 1 targeting the Virtex 6 device shows a decrease in performance for Batch Elimination while and optimal result for Iterative Elimination. Orthogonal Array optimization increases throughput to area ratio by 7.1%, just 2% below the optimal performance increase. The failing of the Batch Elimination algorithm in this case can be attributed to interaction effects.

Experiment 2 targeting the Virtex 6 device shows very poor performance for Batch Elimination, achieving a decrease in 7% with respect to the original baseline.

Iterative Elimination with the larger option set leads to identical performance as realized with the smaller option space. The options added in the larger option set have no effect on the design optimization in this configuration. Frequency Search leads to a performance increase of 10.2%. Applying Batch Elimination after the frequency search decreases overall performance but does have a performance increase of 3.5% over the Least Effort optimization. The poor performance of Batch Elimination was expected due to the prevalence of interaction effects in previous applications of Batch Elimination for this design target device pairing. Iterative Elimination applied after Frequency Search increases performance by an additional 6% leading to an overall performance increase of 16.7%. Placement Search applied following the Batch Elimination Frequency Search chain yields a total performance increase of 18.8% outperforming the old ATHENa environment by nearly 4%. The final optimized result for this design device target pairing is 21.1% above the Least Effort optimization performance. This result is also nearly 7% higher than the best result achieved by the old ATHENa environment.

Table 9 Skein Results

	Algorithm	**Area (Slices)**	**Throughput (Mbits/s)**	**T/A**	**% Impr.**	**#Runs**	**Area (Slices)**	**Throughput (Mbits/s)**	**T/A**	**% Impr.**	**#Runs**
		Spartan 3					**Virtex 6**				
Experiment 1	LE	3199	416	0.130	0.0	1	914	1232.202	1.348	0.0	1
	ME	2447	376	0.154	18.4	32	836	1232.854	1.475	9.4	32
	BE	3100	416	0.134	3.3	7(2)	905	1187.959	1.313	-2.6	7(2)
	IE	3045	419	0.138	5.9	11(2)	836	1232.854	1.475	9.4	18(4)
	OA	3317	422	0.127	-1.9	9(2)	836	1207.801	1.445	7.2	9(2)
Experiment 2	LE9	2944	420	0.143	9.7	1	959	1202.008	1.253	3.9	1
	BE9	2870	417	0.145	11.9	11(2)	922	1144.348	1.241	-7.9	11(2)
	IE9	2886	425	0.147	13.3	34(4)	836	1232.854	1.475	9.4	34(4)

	FS	2906	439	0.151	16.4	21(4)	969	1439.891	1.486	10.2	21(4)
	FS=>BE9	2874	431	0.150	15.4	32(6)	890	1242.245	1.396	3.5	32(6)
	FS=>IE9	2874	431	0.150	15.4	31(5)	882	1388.579	1.574	16.8	61(9)
	FS=>BE9=>PL	2874	431	0.150	15.4	53(8)	918	1470.380	1.602	18.8	53(8)
	FS=>IE9=>PL	2874	431	0.150	15.4	52(7)	867	1415.764	1.633	21.1	82(11)
	GMU-1	2870	417	0.145	11.9	--------	858	1330	1.55	14.9	--------

The results for the Keccak design targeting the Spartan 3 device show a decrease in performance of 1.3% for Batch Elimination in Experiment 1. Iterative Elimination reaches the optimal solution in the confined search space of 10.78% improvement and the Orthogonal Array algorithm leads to an near optimal improvement of 8.47%.

Spartan 3 Experiment 2 results show an improvement of 12.5% and 14.7 % for Batch Elimination and Iterative Elimination respectively. Frequency Search provides an improvement of 13.3% to the baseline results. Batch Elimination applied after the frequency search lowers the performance increase to only 4.39%, indicating a potential issue when applying optimizations with the Frequency Search in this way. Iterative Elimination increases performance to 19.13% after the frequency search. After applying Placement Search following the Frequency Search, Batch Elimination optimization chain, performance reaches 13.63% improved over the baseline. The application of a placement search following the Iterative Elimination optimization chain leads to no further improvement. The fully optimized result achieves a 10% performance advantage over the old ATHENa environment in a low 61 serial runs, and only a parallel runtime of 8 parallel runs.

For Experiment 1 targeting the Virtex 6 device Batch Elimination shows a decrease in performance of 2.5%. An improvement of only 1.1% is shown by using the Iterative Elimination algorithm. The Orthogonal Array optimization also shows a decrease in performance registering a decrease of 3.7%. The Keccak design appears to have more interaction effects than most designs when targeting the Virtex 6 device. Another key issue in this experiment was that the maximum performance increase was quite low, only 6.4% showing that the design was harder to optimize than some.

For Experiment 2 Batch Elimination leads to a decrease of 3.4% while Iterative Elimination leads to an increase in performance of 6.9%. Frequency Search yields an increase of 26.3% with respect to throughput to area ratio. The additional application of Batch Elimination leads to significant performance decrease with a relative improvement of -27%. Iterative Elimination outperforms the baseline by 22%, but less than the 26% achieved by Frequency Search alone. Placement search has no effect on the Batch Elimination results. When Placement Search is applied to Iterative Elimination the performance is increased to a total of 24.4% above the baseline.

Table 10Keccak Results

	Algorithm	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs	Area (Slices)	Throughput (Mbits/s)	T/A	% Impr.	#Runs
		Spartan 3					Virtex 6				
Experiment 1	LE	3398	4641	1.37	0.0	1	1132	9928	8.770	0.0	1
	ME	3062	4633	1.51	10.8	32	1068	9972	9.337	6.5	32
	BE	3441	4637	1.35	-1.3	7 (2)	1162	9930	8.546	-2.6	7 (2)
	IE	3062	4633	1.51	10.8	15 (3)	1144	10143	8.867	1.1	15 (3)
	OA	3317	4641	1.40	8.5	9 (2)	1215	10258	8.443	-3.7	9 (2)
r i m	LE9	3129	4636	1.48	6.9	1	1221	9757	7.991	-8.9	1

	BE9	3175	4636	1.46	12.5	11 (2)	1154	9776	8.472	-3.4	11 (2)
	IE9	2971	4654	1.57	14.7	34 (4)	1173	11000	9.378	6.9	34 (4)
	FS	3135	4851	1.55	13.3	21 (4)	1276	14144	11.09	26.4	21 (4)
	FS=>BE9	3408	4859	1.43	4.4	32 (6)	1355	10876	8.027	-27.6	32 (6)
	FS=>IE9	2990	4865	1.63	19.1	40 (6)	1256	13436	10.69	22.0	55(8)
	FS=>BE9=>PL	3129	4856	1.55	13.6	53 (8)	1355	10876	8.027	-27.6	53 (8)
	FS=>IE9=>PL	2990	4865	1.63	19.1	61 (8)	1129	12352	10.94	24.4	76(10)
	GMU-1	3366	5039	1.50	9.8	---------	1086	11839	10.90	24.3	---------

Experiment 1 was intended to explore the ability of the optimization space exploration algorithms to achieve optimal performance in a constrained option space. The goal was to see how well the algorithms optimized the designs and if they were a good candidate for replacing Most Effort optimization for the enhanced ATHENa environment.

Batch Elimination was the first algorithm tested and was the least resource intensive algorithm of the three candidates. For all the designs tested and both targeted FPGA devices Batch Elimination never reaches the optimal result. On average Batch Elimination achieves 84% less performance than the optimal option combination with a median of 79% below optimal for the Spartan 3 device. Targeting the Virtex 6 device results are similar showing an average underperformance of 82% with a median of 82%. This underperformance can be attributed to the largest drawback of the Batch Elimination algorithm, the inability to handle interaction effects. The Xilinx FPGA tool options that were used to optimize the cryptographic has algorithms presented here exhibited some mild to extreme interaction effects in each case. These effects are completely ignored by the Batch Elimination algorithm and lead to the observed poor performance. While performance of the Batch Elimination algorithm was poor relative to the optimal solution, it did outperform the Least Effort optimization by an average of 3.7% and 7.4% for Spartan 3 and Virtex 6 device conditions respectively. The 9 option version of the algorithm performed marginally better achieving an average performance increase of 10% for Spartan 3 and 3% for Virtex 6. The results of the 9 option version is not surprising given that the performance could increase due to more options being investigated but more interaction effects could also be introduced.

Iterative Elimination performed the best of the three optimization space exploration algorithms introduced. Due to the iterative nature of the algorithm it was able to mitigate the effects of simple interaction effects and achieve good performance in most cases. In tests targeting the Spartan 3 device the Iterative Elimination algorithm averaged only 18% below the optimal performance with a median of only 2% below. For two of the four hash algorithms Iterative Elimination achieved the optimal result and underperformed by 4.7% in a third. When applied to the Skein has algorithm Iterative Elimination underperformed the optimal result by 67%. It should be noted that when targeting the Spartan 3 device, all of the optimization space exploration algorithms severely underperformed Most Effort optimization. This can again be attributed to stronger interaction effects between the options for this design device pairing. Iterative Elimination when targeting the Virtex 6 device achieves the optimal option set in 3 of the 4 tests. The average that it underperformed the optimal solution was nearly 20% due to underperforming by 83% for the Keccak design. Like the Spartan targeted optimizations before it the one design that caused Iterative Elimination issues also exhibited the same strong interaction effects for all other algorithms as well. The performance gain realized when using Iterative Elimination was 16% and 15% for the Spartan 3 and Virtex 6 devices respectively. The 9 option version achieved a 20% average performance increase for the Spartan 3 device and a 16% performance increase for the Virtex 6 device. The performance of the 5 option version is almost identical to the 9 option version for the Virtex 6 device, differing only when optimizing the Keccak design. This behavior can be attributed to the fact that the options that drove the optimization most were the options common to both option sets. This behavior does not show itself for the Batch Elimination algorithm due to the limits imposed on that algorithm by even simple interaction effects.

The Orthogonal Array algorithm proved to result in performance between the Batch Elimination algorithm and Iterative Elimination algorithm in most cases. Targeting the Spartan 3 device the Orthogonal Array algorithm underperformed the optimal results by an average of 62%. The algorithm does reach only 7% below optimal for the JH design. With respect to the optimizations performed targeting the Virtex 6 device the Orthogonal Array algorithm underperforms the optimal result by 52% but with a median of only 25% achieving the optimal result for the JH design. While the Orthogonal Array algorithm is still susceptible to interaction effects it is less influenced by them than the Batch Elimination algorithm. If a stronger underlying orthogonal array was used to construct the algorithm interaction effects could in theory be further mitigated. The Orthogonal Array algorithm achieves a nearly 5% and 11% performance increase relative to Least Effort optimization for the Spartan 3 and Virtex 6 devices respectfully. The largest drawbacks of the Orthogonal Array algorithm is the need to construct an entirely new orthogonal array for different sized option sets and the inability of the algorithm to optimize options that take on more than two states.

Table 11 Performance Relative to Most Effort Spartan 3

	JH	BLAKE	Skein	Keccak	Avg. %below opt	Med. %below opt
BE	-68.5	-76.1	-82.3	-112.4	-84.8	-79.2
IE	-4.8	0.00	-67.7	0.0	-18.1	-2.4
OA	-7.7	-109.2	-110.8	-21.4	-62.3	-65.3

Table 12 Performance Relative to Most Effort Virtex 6

	JH	BLAKE	Skein	Keccak	Avg. %below opt	Med. %below opt
BE	-36.3	-27.5	-128.0	-139.5	-82.8	-82.2
IE	0.0	0.0	0.0	-83.2	-20.8	0.0
OA	0.0	-27.2	-23.7	-157.5	-52.1	-25.4

Table 13 Performance Relative to Least Effort Spartan 3

	ME	BE	IE	OA	BE9	IE9
JH	16.8	5.3	16.0	15.5	11.4	21.9
BLAKE	33.0	7.9	33.0	-3.0	7.9	33.0
Skein	18.4	3.3	5.9	-1.9	11.9	13.3
Keccak	10.8	-1.3	10.8	8.5	12.5	14.7
Average %inc	19.8	3.8	16.4	4.7	10.9	20.7
Median %inc	17.6	4.3	13.4	3.2	11.6	18.3

Table 14 Performance Relative to Least Effort Virtex 6

	ME	BE	IE	OA	BE9	IE9
JH	13.5	8.6	13.5	13.5	8.6	13.5
BLAKE	36.4	26.4	36.4	26.5	15.1	36.4
Skein	9.4	-2.6	9.4	7.2	-7.9	9.4
Keccak	6.5	-2.6	1.1	-3.7	-3.4	6.9
Average %inc	16.4	7.5	15.1	10.9	3.1	16.6
Median %inc	11.4	3.0	11.4	10.3	2.6	11.4

The enhanced ATHENa environment proved more than capable of outperforming the old ATHENa environment in all but one case investigated. When targeting the Spartan 3 device the new environment outperformed the legacy environment by an average of nearly 16%, achieving notable gains of 31% for the BLAKE design and 19% for the JH design. The results for the Virtex 6 device target was much closer, underperforming the old environment by an average of 2.3%. This result is misleading, skewed by the fact that the old environment outperformed the new one by 16% for the JH design but underperformed in every other design.

The value added with the new ATHENa environment goes beyond just the performance achieved but also encompasses the optimization time needed to complete the benchmarking tasks. For the experiments conducted none of the fully optimized designs took longer than 13 parallel runs, easily realized using just two multicore workstations. Due to the parallel nature of the algorithms presented, if more compute resources were available more options could be investigate, or finer resolution Frequency and Placement Searches could be performed, leading to even better performance in theory. Table 15 shows the maximum performance achieved by the new and old ATHENa for each targeted device. Figure 9 illustrates the performance of the systems targeting the Spartan 3 device while Figure 10 illustrates the performance of the systems targeting the Virtex 6 device.

Table 15 Maximum Performance

	Saprtan3	Virtex 6	S3 GMU	V6 GMU
Keccak	19.1	26.5	9.8	24.3
JH	42.5	54.4	23.1	70.7
Skein	16.4	21.1	11.9	14.9
Blake	33.0	44.3	2.5	43.5
Average %inc.	27.8	36.6	11.8	38.4
Median	26.1	35.4	10.9	33.9

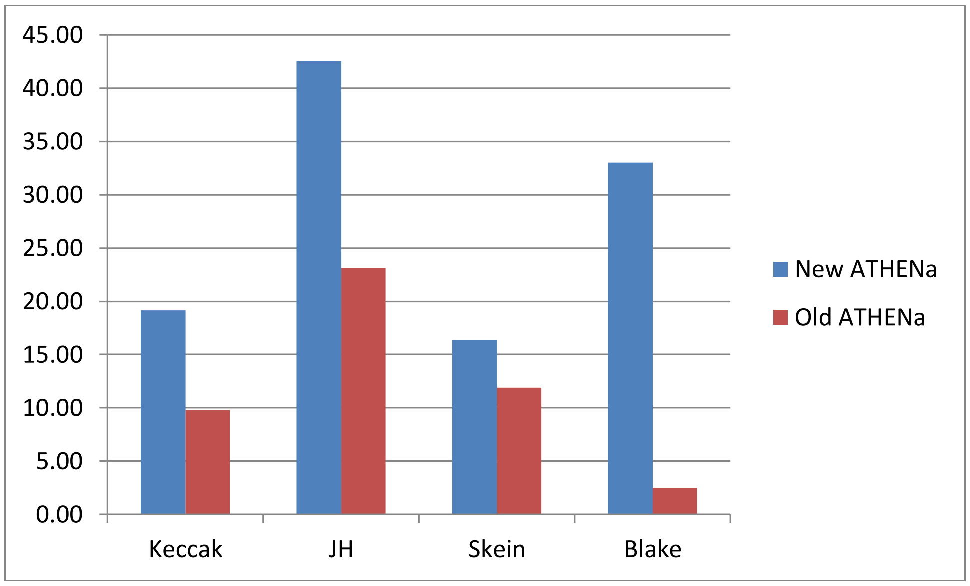


Figure 9 New ATHENa vs. Old ATHENa: Max Performance Spartan 3

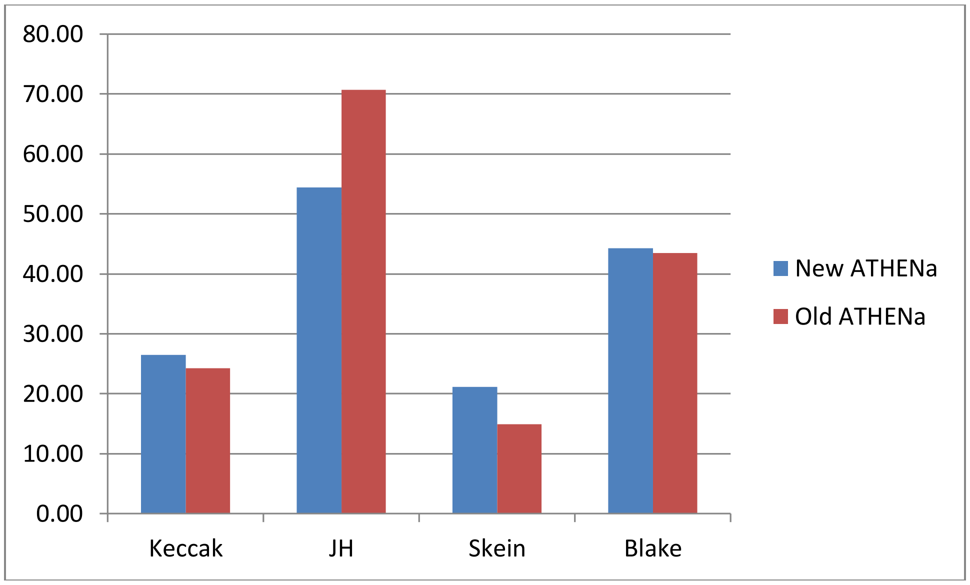


Figure 10 New ATHENa vs. Old ATHENa: Max Performance Virtex 6

Algorithm chaining using the Frequency and Placement Search algorithms in conjunction with Iterative Elimination and Batch Elimination yielded the most optimized result in most cases. While these results are quite promising additional work needs to be done to determine the optimal way to chain the algorithms together. Different results can be achieved by running the optimization space exploration first and then executing the search algorithms. It is also possible to run a preliminary Placement Search as one of the options investigated in the optimization space exploration algorithms, resulting in yet a different optimized design. The difficulty in determining how to chain the search algorithms and optimization algorithms together is due the dependencies of each stage with the other. By using a different option set as input to the Frequency Search, a different result is obtained. The corollary is also true, by adjusting the input frequency used by Batch Elimination and Iterative Elimination, a different option set could result, and most often does. These dependencies have not been fully explored yet and could lead to better insight into how these algorithms could be used together to greatest effect.

9. CONCLUSION

Through the use of the ATHENa benchmarking and comparison environment extended with a distributed batch system and the addition of new optimization space exploration algorithms the efficiency of benchmarking and comparison tasks can be greatly increased.

Using the Condor distributed batch system allows for the scaling of the ATHENa environment to utilize as many compute nodes as desired. By increasing the number of benchmarking subtasks that can be executed in parallel the time to complete most benchmarking tasks decreases significantly. While decreased execution time is a welcome addition it also aids in more thorough benchmarking and optimization of candidate algorithms by allowing users to test and optimize using more options than feasible in a serial or multiprocessor system. Condor is especially suited to this task allowing heterogeneous compute resources to be pooled into a high throughput computing cluster. This along with other features of Condor allow users of the system to utilize any available compute resource that supports their desired design tools, even fairly sharing resources that are owned and used by other users.

In addition to adopting the Condor distributed batch system, a graphical front-end was developed for ATHENa to aid in monitoring and managing the tasks executing on the system. By giving a user a better picture of the available resources and control over the execution of the jobs executing on these resources users can more easily and efficiently use the system. The interface allows users to view all of the current compute nodes available, view all running and queued tasks, and run, pause, resume and cancel any task in the system. Beyond the monitoring and management features a job submission wizard is also included to aid in the creation of new job submissions. The wizard allows a user to build a custom job submission from pre-defined building blocks that include tool option sets, optimization algorithms, optimization targets and algorithms. The addition of the graphical front-end lowers the learning curve for users of the system and increases the overall usability of ATHENa.

The last addition to the ATHENa environment and the focus of most of the research are the new optimization space exploration algorithms. In total five new algorithms were designed for use with the enhanced ATHENa environment. Frequency Search and Placement Search are parallel implementations of two searches that existed in the original ATHENa. By increasing the parallelism of these two searches both become much more efficient as a part of the benchmarking toolset. Frequency Search can be accomplished in four parallel steps, while Placement Search can be accomplished in just two. While this is impressive it must be noted that at least eleven compute resources must be available to achieve these results. If run on a single compute resource both searches require twenty one sequential runs to complete.

Batch Elimination was the first completely new algorithm introduced and was shown to achieve moderate success as an optimization algorithm for the ATHENa environment. The algorithm can be completed in as few as two parallel steps with *n* resources where *n* is the number of options investigated in the optimization. If only one compute resource is available Batch Elimination can complete in *n+2* sequential runs. This balance of parallel and sequential runtime makes Batch Elimination a good algorithm to use in a resource constrained environment. While Batch Elimination does not guarantee near optimal results, the results are often better than a naïve Least Effort optimization. It is also a good choice for when it is unclear what state an option should take on for a particular optimization target. The largest drawback of Batch Elimination is the inability of the algorithm to handle what are known as interaction effects, or when an option changes behavior in the presence of another option or set of options. Batch Elimination only tests options in isolation so interaction effects are unaccounted for. Because of the interaction effect liability of Batch Elimination it is recommended for use with fewer options or options known not to interact so as to mitigate these effects.

Iterative Elimination is a direct evolution of the Batch Elimination algorithm modified to handle basic interaction effects. Iterative Elimination chooses only one option per step and then adds that option to the baseline used for the next step. This allows each option to be considered with respect to every option chosen in previous steps. The removal of interaction effects makes Iterative Elimination the most effective optimization algorithm added to the ATHENa environment. In almost all cases Iterative Elimination reaches an optimal or near optimal option set. While the performance of Iterative Elimination with respect to optimization is the best, the runtime is the slowest of the algorithms presented. Iterative Elimination requires at least two parallel runs of the system to complete but can require as many as *n* where *n* is the number of options being investigated for the optimization. Rarely does the algorithm require the worst case runtime, in fact it was never shown to require more than seven parallel runs in the experiments conducted and required less than 4 on average. The largest drawback of the algorithm is the sequential execution time if only one compute resource is available. If required to run serially, the minimum runs required increases to $n+n-1$. The maximum runs required increases to $n * \frac{n}{2} + \frac{n}{2}$. This serial execution time means that it is unsuited for single resource or low resource environments if the number of options investigated is too large.

The Orthogonal Array optimization algorithm is a departure from the two previously described algorithms. It utilizes a mathematical construct called an orthogonal array to determine option sets for each run in an optimization. The orthogonal array has properties that when used to determine the option states of options in an optimization it can approximate a full Most Effort optimization. The Orthogonal Array algorithm as implemented for ATHENa requires only two parallel runs to complete, and at most nine runs in serial striking a good sequential parallel runtime balance. As shown in the experimental results the Orthogonal Array algorithm was the least consistent algorithm providing optimal and near optimal results in a few cases while also underperforming Least Effort optimization in a few cases. This variation in results can be attributed to the fact that a strength two orthogonal array was used for this implementation. The higher the strength orthogonal array the better approximation of a Most Effort optimization and thus optimal result it achieves. This better optimization performance comes at a cost however, increasing the number of runs required for optimization and requiring a much more complex underlying orthogonal array. The main drawback of this optimization algorithm is the fact that it cannot be adapted easily to handle options that have more than two states. This severely limits the usefulness of the algorithm for more comprehensive benchmarking and comparison tasks.

It was shown in the experimental results that by chaining together optimization algorithms like Iterative Elimination, Frequency Search, and Placement Search together that optimization performance can be increased greatly. In almost every case the most optimized result from the enhanced ATHENa environment outperformed the results generated by the old ATHENa environment utilizing the GMU_Optimization_1 optimization technique. While the enhanced ATHENa did require more sequential runs to achieve this improved performance, the parallel runtime utilizing sixteen compute resources realized as two eight core workstations was far less. While chaining optimization techniques proved beneficial, more work and research needs to be done to determine the best way to chain the optimization techniques together. The method used in the experiments sequentially applied Frequency Search followed by either Batch Elimination or Iterative Elimination followed by Placement Search. Vastly different results can be achieved by rearranging the order of the algorithms. It should also be noted that the Frequency Search and Placement Search algorithms presented only work for the Xilinx tool chain and would need to be adapted to work properly for Altera design tools or other tool chains.

While the new algorithms and distributed batch system greatly increased the efficiency and optimization performance of ATHENa, the enhanced environment can be deployed in such a way that it is backwards compatible with the old ATHENa environment. Job submissions can be generated that can run benchmarks with the old ATHENa scripts on an execute host. This is invaluable for verifying old results and utilizing the more serial runtime friendly optimization algorithms developed for the old environment.

It was shown through the four case studies that the enhanced ATHENa environment can greatly aid in the benchmarking and comparison of algorithms implemented in hardware description languages and targeted to FPGAs. The parallelism offered by the enhanced environment decreases the overall runtime and allows larger option spaces to be explored leading to more thorough benchmarking of algorithms. The optimization space exploration algorithms further aid the benchmarking process by allowing near optimal optimization of the algorithms in the presence of large option search spaces where a Most Effort optimization is infeasible. By increasing both the efficiency and performance of ATHENa while also adding features to increase usability the enhanced ATHENa environment is positioned to effectively increase the quality of benchmarking and comparison tasks that are of great interest to the cryptographic engineering community as algorithm competitions gain widespread acceptance and use.

APPENDIX A: CONFIGURATION FILES

Condor Submit File

```
Executable = ATHENaLauncher.sh
Universe = vanilla
Output = ATHENa.$(cluster).out
Log = ATHENa.log
Error = ATHENa.$(cluster).err
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.0"
transfer_input_files = design.config.0.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.1"
transfer_input_files = design.config.1.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.2"
transfer_input_files = design.config.2.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.3"
transfer_input_files = design.config.3.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
```

```
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.4"
transfer_input_files = design.config.4.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.5"
transfer_input_files = design.config.5.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.6"
transfer_input_files = design.config.6.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.7"
transfer_input_files = design.config.7.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.8"
transfer_input_files = design.config.8.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
environment = "Process=$(Process) Cluster=$(cluster)
Name=options.FPLEXLES_balancedBaseName=options.FPLEXLES_balancedNameDC
=design.config.9"
transfer_input_files = design.config.9.txt, options.FPLEXLES_balanced.txt,
rename_workspace.py, single_run.txt
arguments = -new
Queue
```

ATHENa Design Config

```
# ============================================
# ============================================
# Global Settings
# ============================================
# ============================================
[Global Settings]
# work directory, used as a root for all result directories

WORK_DIR = 140_2

# directory containing synthesizable source files for the project
SOURCE_DIR = examples/keccak

# A file list containing list of files in the order suitable for synthesis and implementation
# low level modules first, top level entity last
SOURCE_LIST_FILE = source_list.txt

# project name
# it will be used in the names of result directories
PROJECT_NAME = keccak


# ============================================
# ============================================
# Synthesis and Implementation Settings
# ============================================
# ============================================
[Synthesis and Implementation Settings]
# name of top level entity
TOP_LEVEL_ENTITY =  keccak_top_ppl

# name of top level architecture
#TOP_LEVEL_ARCH = main
TOP_LEVEL_ARCH = structure

# name of clock net
CLOCK_NET = clk

#formula for latency
LATENCY = 15 * TCLK

#formula for throughput
```

```
THROUGHPUT =1088 /(24*TCLK)

# OPTIMIZATION_TARGET = speed | area | balanced
OPTIMIZATION_TARGET = balanced

OPTIONS = FPLEXLES


APPLICATION = exhaustive_search

TRIM_MODE = zip

# ============================================================
# ============================================================

[spartan3]
req_syn_freq=102
req_imp_freq=102
FPGA_DEVICES=xc3s1000fg676-5
```

ATHENa Option File

```
[DESIGN SOFTWARE]
XILINX_SYNTHESIS_TOOL = XST
[XILINX DESIGN SOFTWARE OPTIONS]
xilinx_par_opt = -ol[high] -xe[c]
xilinx_trace_opt = -v[3] -a[]
xilinx_map_opt = -logic_opt[on] -ol[high] -xe[c] -pr[b] -timing[]
xilinx_xst_opt = -opt_level[2] -opt_mode[speed] -netlist_hierarchy[rebuilt] -
cross_clock_analysis[yes]
xilinx_synplify_opt = []
xilinx_ngdbuild_opt = []
```

APPENDIX B: USERS GUIDE

This is a simple user's guide to the enhanced ATHENa benchmarking tool. This guide assumes that the requisite software is already installed on the system and that Condor is configured properly. A separate manual describes the setup and installation of the entire enhanced ATHENa in more detail.

GUI

The first step from a user's perspective is to launch the ATHENa GUI. This is done by typing: "python Submitter.py" from the command line in the directory that the ATHENa client application resides in. Once the GUI has been launched use the mouse to click the bottom icon on the set of right hand icons.

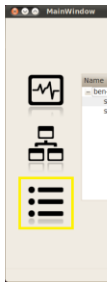
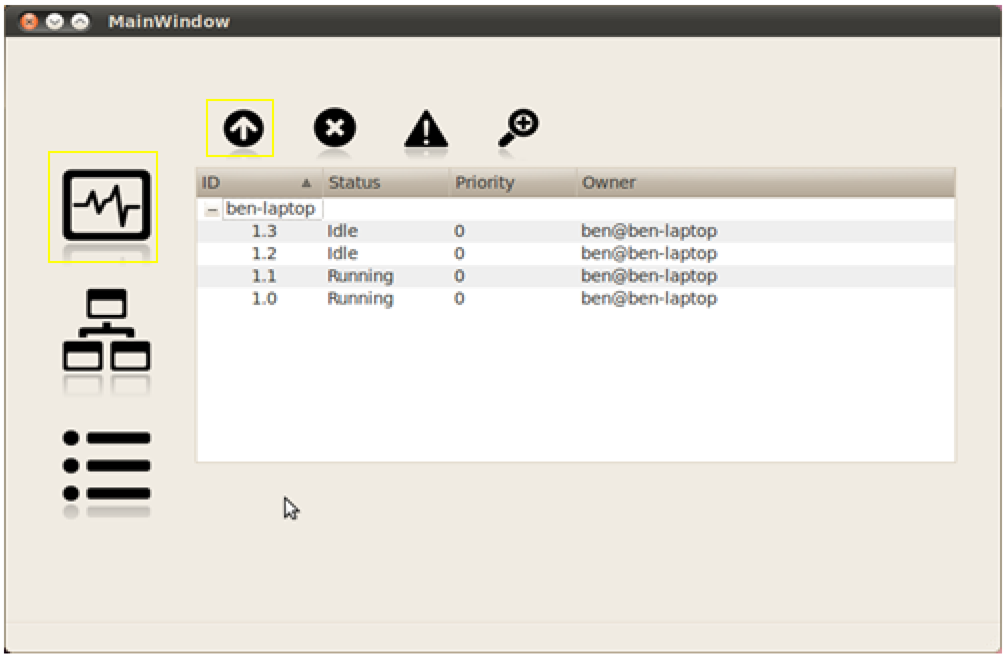


Figure 11 Configuration Wizard Icon

This launches the Configuration Wizard. The first page of the configuration wizard contains fields to set the name of the project, the source list file, the source directory and the work directory for the benchmarking task that you are configuring. Subsequent pages of the Wizard contain various settings for the project. When configuring the benchmarking tasks with the Configuration Wizard there are still some manual step involved at this time. When selecting the application that is to be executed there are additional configuration files that need to be configured and used that are not controlled by the Configuration Wizard. If the application selected is single_run, an option file representing the tool options to be used must be created with the name "options.[OPTIONS]_[OPTIMIZATION_TARGET}.txt" where [OPTIONS] and [OPTIMIZATION_TARGET] are replaced by the selections from the Configuration Wizard for those fields. The end result should be a file named something like "options.default_balanced.txt". The format of the file should follow the format in Appendix A for ATHENa Options files. If the application is any of placement_search, exhaustive_search, batch_elimination, iterative_elimination, OA or frequency_search, then two files need to be edited and configured. The first file that needs to be configured is a file named "exhaustive_search.txt". This file contains two entries, EXHAUSTIVE_SEARCH_STRATEGY, and TYPE. The EXHAUSTIVE_SEARCH_STRATEGY should be set to an ATHENa Option filename that you wish to use for the optimization being used. The TYPE should be set to the name of the optimization you wish to execute. To run an exhaustive search, TYPE should be set to all. For all others it should be set to the names specified previously. As with the single_run application the ATHENa Option file needs to be configured as shown in Appendix A.

Once a configuration has been created and saved, it is now time to execute the configuration. This is done by clicking the submissions icon from the left and then clicking the submit icon from the top row as shown in the diagram below.



ID	Status	Priority	Owner
– ben-laptop			
1.3	Idle	0	ben@ben-laptop
1.2	Idle	0	ben@ben-laptop
1.1	Running	0	ben@ben-laptop
1.0	Running	0	ben@ben-laptop

Figure 12 Job Submission Process

Once the submission icon is pressed a dialog appears that browses the local directory. Select the newly created configuration file from the file listing and the Client will extract submit the job for processing. When this occurs a directory will be created that corresponds to the major job id denoted as “run_[major job id]/”. This directory will contain the results of the job when the job completes. Two files will be contained within this directory that contain the options used and the results, option_summary.csv, and report_summary.csv respectively.

REFERENCES

REFERENCES

[1] K. Gaj, J. Kaps, V.Amirineni, M. Rogawski, E.Homsirikamol, B. Brewster, *"ATHENa – Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware using FPGAs,"* 2010 International Conference on Field Programmable Logic and Applications, 2010.

[2] D. Bernstein and T. Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to, accessed 12 March 2012.

[3] D. Bernstein and T. Lange, *"ECRYPT Benchmarking of Cryptographic Systems,"* CHES 2009 Workshop Benchmarking Cryptographic Hardware, 2009

[4] S.Triantafyllis, M.Vachharajani, N. Vachharajani, D. I. August, *"Compiler Optimization Space Exploration,"* International Symposium on Code Generation and Optimization, 2003.

[5] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience" *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[6] University of Wisconsin Computer Science, The Condor Project. http://research.cs.wisc.edu/condor/ (accessed March 21, 2012).

[7] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", *Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL.

[8] Z. Pan, and R. Eigenmann, *"Fast and effective Orchestration of Compiler Optimizations for Automatic Performance Tuning,"*International Symposium on Code Generation and Optimization, 2006.

[9] R.P.J. Pinkers, P.M.W Knijnenburg, M. Haneda, and H.A.G. Wijshoff, *"Statistical Selection of Compiler Options,"*The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2004.

[10] P.M.W. Knijnenburg, M. Haneda, and H.A.G. Wijshoff, "Automatic Selection of Compiler Options using Non-Parametric Inferential Statistics,"14th International Conference on Parallel Architectures and Compilation Techniques, 2005.

[11] K.Gaj, E. Homsirikamol, M. Rogawski, R. Shahid, and M. Sharif, *"Comprehensive Evaluation of High-Speed and Medium Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs,"*The 3rd SHA-3 Candidate Conference, Washington, D.C., March 22-23, 2012.

[12] Cryptographic Engineering Research Group, GMU Source Codes. http://cryptography.gmu.edu/athena/index.php?id=source_codes (accessed March 4, 2012).

CURRICULUM VITAE

Benjamin Brewster graduated from Centreville School, Centreville, Virginia, in 2001. He received his Bachelor of Science from West Virginia University in 2005. He is employed as a software engineer with seven years of experience in the defense industry.