Domain-specific Accelerators for Ideal Lattice-based Public Key Protocols

The case of NewHope and BLISS-BI

Hamid Nejatollahi[1], Nikil Dutt[1], Indranil Banerjee[2] and Rosario Cammarota[2]

[1] University of California Irvine, Irvine, USA
[2] Qualcomm Technologies Inc., San Diego, USA
{hnejatol,dutt}@ics.uci.edu, {ibanerje,ro.c}@qti.qualcomm.com

Abstract. Post Quantum Lattice-Based Cryptography (LBC) schemes are increasingly gaining attention in traditional and emerging security problems, such as encryption, digital signature, key exchange, homomorphic encryption etc, to address security needs of both short and long-lived devices — due to their foundational properties and ease of implementation. However, LBC schemes induce higher computational demand compared to classic schemes (e.g., DSA, ECDSA) for equivalent security guarantees, making *domain-specific acceleration* a viable option for improving security and favor early adoption of LBC schemes by the semiconductor industry. In this paper, we present a workflow to explore the design space of domain-specific accelerators for LBC schemes, to target a diverse set of host devices, from resource-constrained IoT devices to high-performance computing platforms. We present the first results of design space exploration on workloads executing NewHope and BLISS-BI schemes accelerated by our domain-specific accelerators, with respect to a baseline without acceleration. We show that achieved performance with acceleration makes the execution of NewHope and BLISS-BI comparable to classic key exchange and digital signature schemes while retaining some form of general purpose programmability. In addition to 44% and 67% improvement in energy-delay product (EDP), we enhance performance (cycles) of the sign and verify steps in BLISS-BI schemes by 24% and 47%, respectively. Performance (EDP) improvement of server and client side of NewHope key exchange is improved by 37% and 33% (52% and 48%), demonstrating the practicality of the approach to favor early adoption.

Keywords: Public Key Cryptography, Post-quantum Cryptography; Lattice-based Cryptography; Ideal Lattices; Key Exchange; Digital Signature; System on Chip; Domain Specific Acceleration; Cache Architecture.

1 Introduction

Lattice based cryptographic (LBC) schemes promise to provide quantum resistance, versatility and ease of implementation compared to other post-quantum cryptography families. LBC primitives are versatile, as they can be used not only for both public key and symmetric key cryptography, but also for new applications such as homomorphic encryption. Thus, in addition to offering protection beyond the span of traditional cryptography and against Quantum cryptanalysis [Sho97, Lon01], the foundational properties of LBC and their ease of implementation make LBC primitives powerful and unique candidates for inclusion in future cryptography standards [NDC17]. Indeed, lattice-based cryptography schemes promise to enhance security for long-lived systems (e.g., critical infrastructure), as well as for safety-critical devices (e.g., smart medical implants) [GMRS+15]. As with other post-quantum cryptography families, LBC computational requirements are much larger compared to classic cryptographic primitives in terms of both key size and computational requirements, thus conflicting with the demanding design constraints (speed of execution, area and power consumption) of cryptographic blocks in modern Systems on Chip (SoCs). While custom hardware accelerators have been deployed for standardized cryptography to tackle system design constraints [WWA01], emerging new cryptography schemes, improved variant of the scheme, or successful attacks of a scheme necessitate a complete redesign of whole custom hardware accelerator. The situation for LBC schemes (e.g., for public key cryptography) is further complicated since these use cases are not yet standardized; we are unable to actually test the strength of such primitives (without a quantum computer);

Finally, emerging variants of LBC schemes greatly discourage the use of custom hardware accelerators. For instance since the first appearance of LBC key exchange protocol named NewHope [ADPS15], new and better in some sense, lighter weight variants have appeared, e.g., NewHope-Simple [ADPS16]; and more schemes will continue to appear rapidly, making custom hardware implementations obsolete for each such new scheme. Alternatively, we focus on the design of domain specific accelerators for LBC schemes that support early adoption of emerging LBC schemes, by relaxing some benefits of custom hardware. On one hand, adopting LBC schemes provides post-quantum resistance to data in transit at the cost of accelerator area (thus the importance of early adopting such schemes) [Pei14]. On the other hand, the focus on domain specific accelerators enables some programmable features of general purpose platforms such that the same accelerator can be programmed to use new, stronger, standardized LBC schemes, while being significantly more energy efficient compared to a general purpose processor. In this work, we use gem5-Aladdin [SXS+16], to perform the design space exploration (DSE) of domain specific accelerators as applied to LBC schemes. Storage and bandwidth play crucial role in the implementation of LBC schemes which can be alleviated by hardware approximation [BHS18]. The exploration workflow provides early estimates of area and power consumption, in addition to performance. The early DSE enables identification of design points for a variety of deployment scenarios for LBC schemes accelerators, in the gamut from resource constrained IoT devices (where area and power consumption are paramount), to server class devices (where throughput and power consumption are the main constraints).[1] [2] Once deployed, the programmability feature of the proposed designs enables execution of families of LBC schemes with the same accelerator, thus providing agility at run-time, a desirable feature to tolerate rapid environment changes across use cases and the emergence of new, better LBC schemes. The DSE workflow is based on gem5-Aladdin, a pre-RTL design flow which allows rapid exploration of design points of a template architecture, and emits performance, area and power estimates. This facilitates the rapid investigation of multiple design points before RTL coding, and provides a tool to quickly re-assess performance, and changes in the presence of new cryptographic schemes. In addition, the integration with gem5 enables the estimation of the benefits of designated accelerator designs when integrated within a SoC. The DSE framework adds an analytic front-end to gem5-Aladdin that prunes the number of design points to those that provides energy efficiency advantages when integrated in the SoC .We demonstrate the utility and efficacy of the DSE workflow by presenting design space exploration results and selecting instances of accelerators for two popular LBC schemes for key exchange, NewHope, and digital signature, BLISS-BI. The core framework offers different design template options. Due to the non-bulky data movement and processing for public key cryptography use cases, the exploration framework focuses on a template design of a loosely coupled, cache-based domain specific accelerator, to accelerate portions of a designated workloads. The design space exploration results illustrate design points that can be suitable for their deployment on the client or on the server side of the protocol, and their trade-offs in terms of SoC resources with respect to the host processor. In conjunction with a main host processor, our selected designs for NewHope and Bliss enhance performance of the sign and verify steps in BLISS-BI schemes by 24% and 47%, respectively,with 44% and 67% improvements in energy-delay product (EDP). The performance (EDP) improvement of server and client side of NewHope key exchange are improved by 37% and 33% (52% and 48%) respectively.

The rest of the paper is organized as follows: Section 2 provides the background for LBC primitives and design space exploration; Section 3 provides the characterization of two LBC schemes, NewHope and BLISS-BI; Section 4 describes the design flow; Section 5 discusses the design space exploration for NewHope and Bliss (both client and server side are accounted and accelerated); Section 6 elaborates on design point selection; Section 7 discusses prior art in accelerating LBC schemes in hardware and software; Known limitations of the proposed approach are discussed in Section 8; Concluding remarks are presented in Section 9.

[1]A multimedia server, for example, requires to establish and maintain a large number ($\tilde{1}$0,000 to 40,000) of simultaneous secure channels per hour, which includes many signature operations per seconds.

[2]https://people.freebsd.org/~rrs/asiabsd_2015_tls.pdf

2 Background

2.1 Lattices

Lattice $\mathcal{L}$ is defined as a countable set of points in a n-dimensional Euclidean space with a periodic structure [MR09]. Specifically, let $B = [b_1, b_2, ..., b_n] \in \mathbb{R}^m$ be a set of linearly independent vectors ($m \times n$ matrix), a lattice is defined as: $\mathcal{L}(B) = \{Bx : x \in \mathbb{Z}^n\} = \{\sum_{i=1}^{n} x_i b_i : x_i \in \mathbb{Z}, 1 \leq i \leq n\}$.

Closest vector problem (CVP) and shortest vector problem (SVP), and their variants, are two lattice hard problems that closely coupled to two common average-case lattice-based problems, Learning With Error (LWE) [Reg05] and Shortest Integer Solution (SIS) [Ajt96]. When implemented, standard LBC, e.g., LWE-based, schemes exhibit a relatively large memory footprint due to the large key size (hundreds of kilobyte per public key), which makes implementations of standard LWE-based schemes impractical on constrained devices. In contrast, a different mathematical construct, the *ring-based lattice* (e.g., ideal lattice), is defined over a ring of polynomials $R = \mathbb{Z}_q[x]/(f(x))$, where $f(x) = x^n + f_n x^{n-1} + \ldots + f_1 \in Z[x]$ and R contains all the polynomials with modulo q integer coefficients. [3] The algebraic structure of ring-based lattices not only allows for fast arithmetic, e.g., performing arithmetic computations on numbers in Number-Theoretic Transform (NTT) format which is highly efficient, with a time complexity of $O(n\log n)$ [Nus80], but also allows a reduction in the memory footprint by factor of n. Structured lattices, e.g., Ring-LWE-base schemes, offer key size reduction by a factor of n compared to the standard lattices, e.g., LWE-based schemes [LPR10], making Ring-LWE a better candidate for resource constrained devices, such as Wi-Fi capable devices, including medical implants. Yet, some form of hardware acceleration is needed to make the speed and energy efficiency of LBC protocols, e.g., key exchange and digital signature, comparable to that of classic protocols, e.g., ECDH, ECDSA.

2.2 LBC Primitives Computational Requirements

In this section, we discuss computationally intensive components (*kernels*) widely used in LBC schemes. Among such schemes, we select NewHope (based on Ring-LWE [LPR10]) and BLISS-BI (based on Ring-SIS [SSTX09]) as exemplars of the state-of-the-art LBC key exchange and digital signature mechanism, and design the accelerators around their protocol definition. In our analysis we care to separate the server vs the client part. In general, if we look at ideal LBC primitives, there are two main components, i.e., the modulo linear algebra part, `polynomial arithmetic`, and the `sampling` (secret, noise and the random polynomial) component.

Polynomial arithmetic

Calculation over the matrices (e.g., matrix multiplication) are the main arithmetic bottleneck in the standard lattices schemes; ring-based lattices with the specific structure avoid matrix arithmetic operations and instead perform polynomial arithmetic (e.g., polynomial multiplication). Multiplication of two large polynomials that can be done by Number-Theoretic Transform (NTT) which is one of the most time consuming blocks in LBC schemes with the ring structure.

NTT is a generalization of Fast Fourier Transform (FFT), which is carried out in a finite field instead of complex numbers with time complexity of $O(n\log n)$. In other words, exp(-2πj/N) with n^{th} primitive root of unity ω_n which is defined as the smallest element in the ring that $\omega_n^n = 1$ mod p and $\omega_\mathrm{n}{}^i \neq 1$ mod p for i $\neq$n. The main idea behind this is to use the point value representation instead of the coefficient representation by applying NTT in $O(n\log n)$; thereafter performing point-wise multiplication in $O(n)$ and finally converting the result to coefficient representation by applying Inverse_NTT (INTT) in $O(n\log n)$.

Sampling component

As with any public key based protocol, for digital signature and key exchange mechanism protocols, generation of random-looking, secret and the error terms along with hashing the message are crucial to keep the scheme resistant against adversaries. SHA-3 is the most recent, fast and reliable secure hash algorithm (SHA) standardized by

[3]Module lattices [LS12] fill the gap between standard and ideal lattices. Module-LWE (resp. Module-SIS) lies between LWE (resp. SIS) and Ring-LWE (resp. Ring-SIS) problems.

NIST in 2015 [Dwo15] that can be used for hashing and/or generating the random term. Keccak [BDPA13], a sponge shaped function, is the heart of the SHA3 standard which can be used as the hash function, pseudo-random number generator or stream cipher. SHA-3 can produce arbitrary number of outputs with one of the output lengths of 224, 256, 384, and 512-bit. In the first step, Keccak *absorb*s blocks of the message (x_i) by processing the input and performs the permutation algorithm on it. Thereafter, it can *squeeze* and generate arbitrary number of outputs.

3 NewHope and BLISS-BI Characterization

In this section we provide the workload characterization of two `LBC` schemes for key exchange and digital signature, NewHope and BLISS-BI respectively, with the goal of co-designing accelerators suitable for both client and server part of the protocol. For characterizing NewHope and BLISS-BI, we use Linux Perf on an Intel Core i7-6700 CPU. The system runs Ubuntu 16.04 with Linux kernel 4.4.0 and the gcc compiler 5.4.1. In the following paragraphs, we first introduce the schemes and their main features, and then their profiling results on the chosen platform.

3.1 NewHope Profiling

Key exchange is the process of trading the keys between two parties in the presence of adversaries. If parties use symmetric keys, the same key is shared between them; otherwise, public key of parties should be exchanged. NewHope is a Ring−LWE based unauthenticated key exchange mechanism (`KEX`) that has attracted attention of research and industry communities. Google, for example, released a version of its Chrome browser which can use NewHope as its key exchange protocol [Bra16]. Reconciliation based Ring−LWE KEX schemes generally consist of three steps. In the first step, Alice (server) generates a public uniformly random polynomial a and two secret noise terms (s and e) from the noise distribution and sends ($b = as + e$,a) to Bob (client). In the second step, Bob performs *Encaps* (a,b) function to calculate the secret key and sends back to Alice necessary (reconciliation) information to calculate the secret key. Lastly, Alice calculates the secret key with her previous data and reconciliation information from Bob.

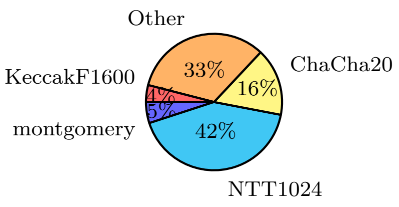
Figure 1 presents the profiling results separated in client and server sides. Cycles in the pie charts are mutually exclusive. Therefore we can already exclude elements on the pie chart from being candidates for acceleration. The number theoretic transform, (used as the `NTT512` for the rest of the paper), consumes most of the cycles in NewHope key exchange mechanism (more than 40%). In order to compute the remainder inside the `NTT`, *Barret* and *Montgomery* reduction techniques are employed.

In order to generate the uniform polynomial $\hat{a}$, the `SHA3-256` secure hash function produces a *seed* (32-byte) which is fed to the `SHAKE-128` extendable output function (XOF). `SHAKE-128` iteratively calls the `Keccak-f[1600]`. Based on the profiling results from `perf`, `Keccak-f[1600]` consumes less that 6% of the whole cycles in NewHope key exchange mechanism.

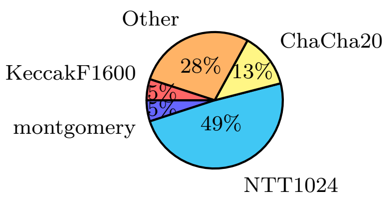
ChaCha20 [Ber08] stream cipher is employed to sample from the centered binomial distribution(ψ_{16}), which consumes 16% and 13% of the total cycles in NewHope scheme for the server and client side, respectively. It turns out that creating an accelerator for ChaCha20 reduces the overall performance of the key exchange mechanism, since the C implementation of ChaCha20 cannot be parallelized enough to compensate the effect of offloading the computation from the fast CPU (2 GHz) to the slow (100 MHz), but parallel, accelerator. In fact, each call (in total 321 calls) to ChaCha20 function consumes less than 80 cycles of the accelerator; hence we do not design the accelerator for ChaCha20. Other functions consume less than 7 percent of the whole application and there are fewer opportunities to speed them up due to their prominent serial fraction; and thus offloading the computation to the accelerator hurts the overall performance of the scheme.

3.2 BLISS-BI Profiling

Digitally signing a document involves sending the signature and document separately. In order to verify the authenticity of the message, the recipient should perform verification on both the signature and the document. A digital signature scheme consists of three steps including key generation, Sign_{sk}, and Verify_{pk}. In the first

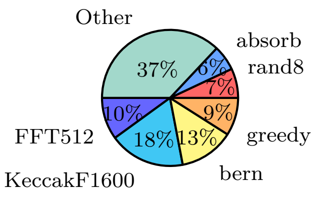


(a) NewHope-Server

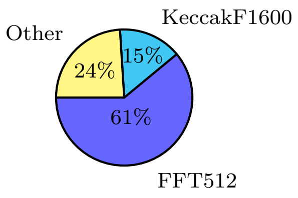


(b) NewHope-Client

Figure 1: Breakdown of NewHope into kernels in terms of (exclusive) CPU cycles (kernels with less than 4% are grouped into *Other*).



(a) BLISS-BI-Sign

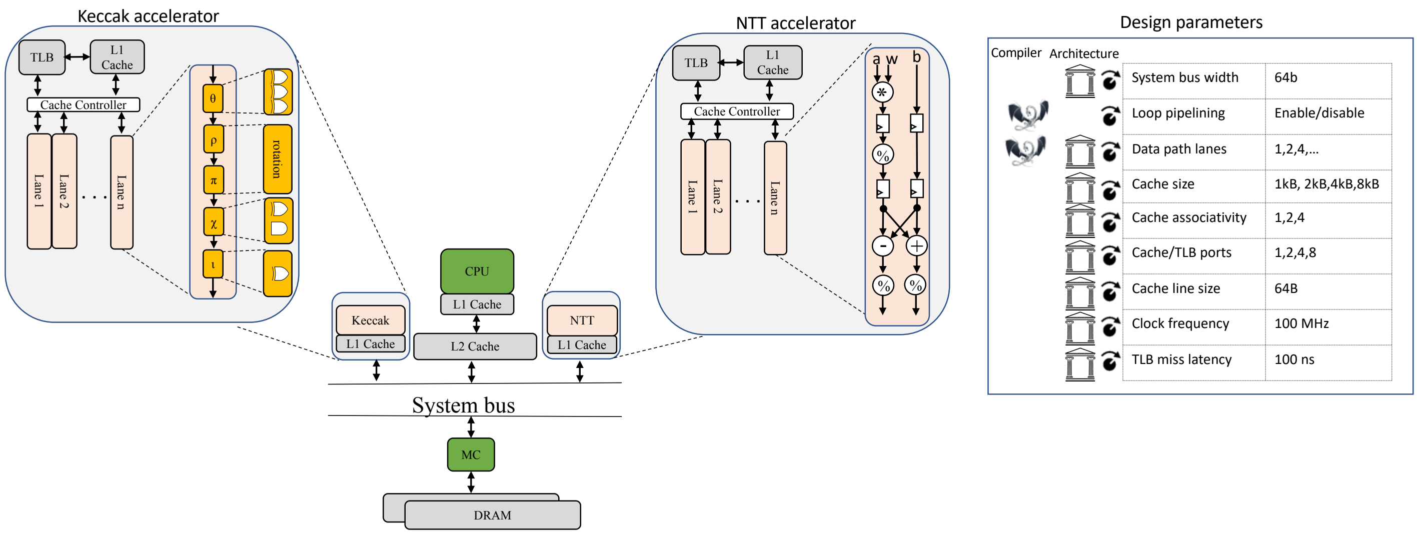


(b) BLISS-BI-Verify

Figure 2: Breakdown of BLISS-BI into kernels in terms of (exclusive) CPU cycles (kernels with less than 4% are grouped into *Other*).

step, secret key (`sk`) and public key (`pk`) are generated; signer keeps the secret key and all verifier parties have the public key of signer. The digital signature S is computed by first hashing the document (M) and then apply a public key operation with the private key on the hash of the document, $S = \text{Sign}_{sk}(H(M), sk_{signer})$. The signer sends the tuple (M,S) to the verifier who applies Verify_{pk} (M,S) and outputs 1 if M and S are a valid message and signature pair; otherwise, S is rejected as the signature of message M. As an example of hash and sign procedure: in the sign step, message M is hashed as $D = h(M)$ where D is the digest. Signer applies the encryption algorithm on D with its private key with the output of $S = Enc(D, sk_{signer})$. Afterwards, signer sends the pair of (M,S) to the verifier. Subsequently, verifier uses public key to decrypt S as $D' = Dec(S, pk_{signer})$. Then verifier compares $D = h(M)$ (same hash function is shared between signer and verifier) and D'; signature is verified if $D' = D$; otherwise, the signature is rejected. The steps outlined for sign and verify are just an example (used in RSA); hence sign and verify steps might be slightly different for various signature schemes, but follow the same idea. Lattice-based signature schemes belong to one of two classes including hash-and-sign (e.g. GPV [GPV08]), and Fiat-Shamir signatures (e.g. BG [BG14], GLP [GLP12], and Lyubashevsky's signature [Lyu09]) [NDR+17]. Bimodal Lattice Signature Scheme (`BLISS`) [DDLL13] is an optimized version of Lyubashevsky's signature where a bimodal Gaussian sampler is used in the rejection sampler component. At the same security level as BLISS, BLISS-B [Duc14] offers up to 2.8× better performance by employing ternary representation of the polynomials which leads random numbers with shorter length.

In this work, we choose BLISS-BI ($n = 512, q = 12289$) as the digital signature for acceleration. Figure 2 provides the profiling results. Polynomial multiplication is performed using NTT (used as `FFT512` for the rest of the paper) which consumes 61% and 10% of the cycles in the verify and sign steps. We use the side-channel resistant implementation of the NTT introduced by Saarinen [Saa17] to perform the polynomial multiplication. To generate the random terms and securely hash the messages, we employ the `SHAKE128` hash function, which invokes `Keccak-f[1600]` function (that consumes 18% and 15% of the cycles in sign and verify steps). Although, Bernoulli sampler (`bern` in Figure 2) consumes 13% of the cycles in the sign step (more than `FFT512`), in order to satisfy the programmability, just `FFT512` and `Keccak-f[1600]` are selected for acceleration. To be more precise, `Keccak-f[1600]` is called inside multiple functions, e.g., `absorb` , `bern`, etc; hence accelerating `Keccak-f[1600]` gives the opportunity to design a single accelerator for both the signer and verifier.



Compiler	Architecture	Parameter	Value
	✓	System bus width	64b
✓		Loop pipelining	Enable/disable
✓	✓	Data path lanes	1,2,4,...
	✓	Cache size	1kB, 2kB,4kB,8kB
	✓	Cache associativity	1,2,4
	✓	Cache/TLB ports	1,2,4,8
	✓	Cache line size	64B
	✓	Clock frequency	100 MHz
	✓	TLB miss latency	100 ns

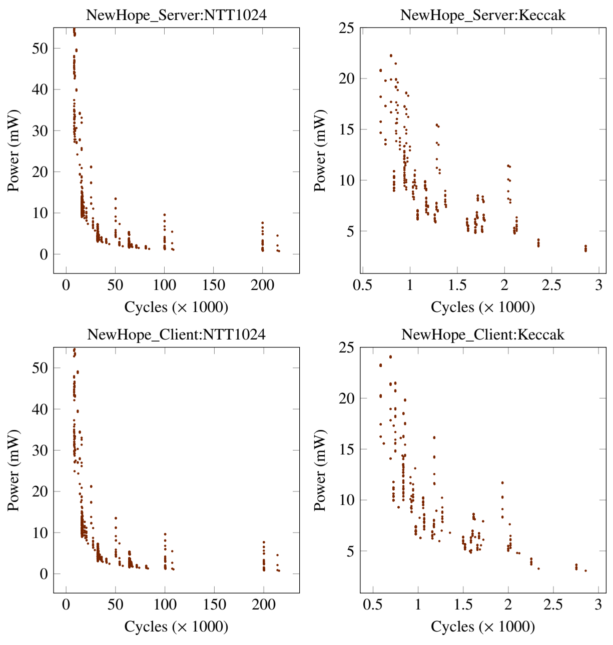
Figure 3: **Architectural template for generated accelerators for `Keccak-f[1600]` and `NTT` kernels.** `NTT1024` and `FFT512` adopt their structure from `NTT`.

4 Design Flow

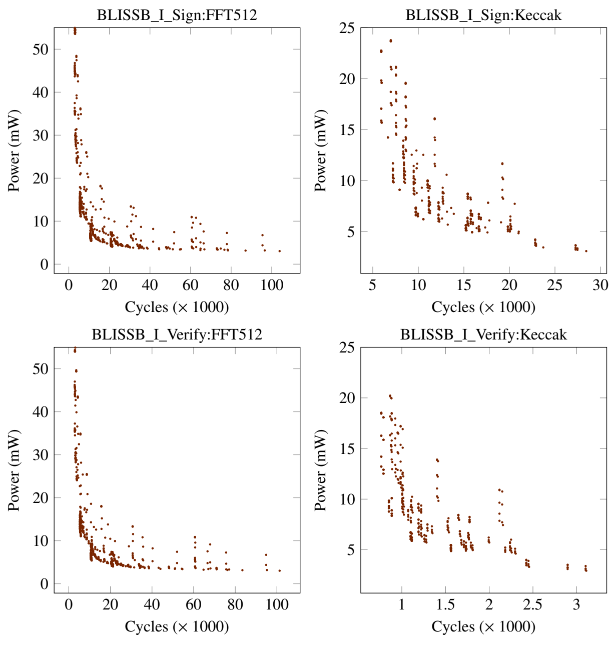
In this section, we provide a brief introduction to gem5-Aladdin SoC simulator [SXS+16] which is the integration of the gem5 system simulator [BBB+11] and Aladdin accelerator simulator [SRWB14]. gem5-Aladdin is a pre-RTL simulation infrastructure that models accelerators and interaction with CPUs in a heterogeneous SoC with CPUs, fixed function accelerators (static datapath accelerator), memory controller and interfaces (DMA engine) that can model performance, area, and power of the accelerator.

After selecting the kernel to accelerate, the programmer annotates the kernel in order to perform design space exploration. gem5-Aladdin takes the annotated C code of the kernel and generates the machine independent intermediate representation (`IR`) of the kernel. Thereafter, it builds the initial dynamic data dependence graph (`DDDG`) based on the dynamic instruction trace that consists of the details of the memory addresses, register ID and opcodes. In the next step, Aladdin performs node-level (bit-width optimization, strength reduction and tree-hight reduction) and loop-level (removes all loop index variable dependencies) and memory (eliminate unnecessary load/store instruction) optimization on the initial DDDG and generates the idealized DDDG that provides flexibility and customization of the datapath of the accelerator. The next step is building the program constrained DDDG in which Aladdin transforms the idealized DDDG into a realistic DDDG by modeling the control and memory dependencies which are missing in the idealized DDDG. Aladdin accounts for the control dependency by adding the non-taken codes into DDDG to model the extra area and power overhead. The idealized DDDG does not account for input dependent memory access; Aladdin alleviates the problem with runtime memory disambiguation. The user can specify hardware resource constraints by compiler and micro-architectural tuning parameters (Figure 3). Compiler level parameters are loop unrolling factor and global loop pipelining. Unrolling factor can be tuned for each loop while the loop (software) pipelining is either active or inactive for all the loops inside the kernel. Micro-architectural tuning parameters include cache size, number of cache ports, cache line size, system bus width, etc. Aladdin estimates the power, area and latency of the accelerator by creating a model based on OpenPDK 45nm that maps each node in the DDDG to a functional unit. SRAM model is built based on CACTI [WJ96].

In order to analyze the results (our core design flow can sweep through 4,000 design points, each in seconds) we developed a visual analytic tool based on Kibana [Sha16], an open source data visualization plugin for Elasticsearch. The tool provides an interactive visualization of the design points which facilitate fast evaluation of the design trade-offs. Figure 9 shows all the design points in a DSE (e.g. FFT512 for BLISS-BI sign step).



(a) NewHope



(b) BLISS-BI

Figure 4: Design space of the accelerators. Each point corresponds to a unique design point by sweeping the design parameters in Figure 3.

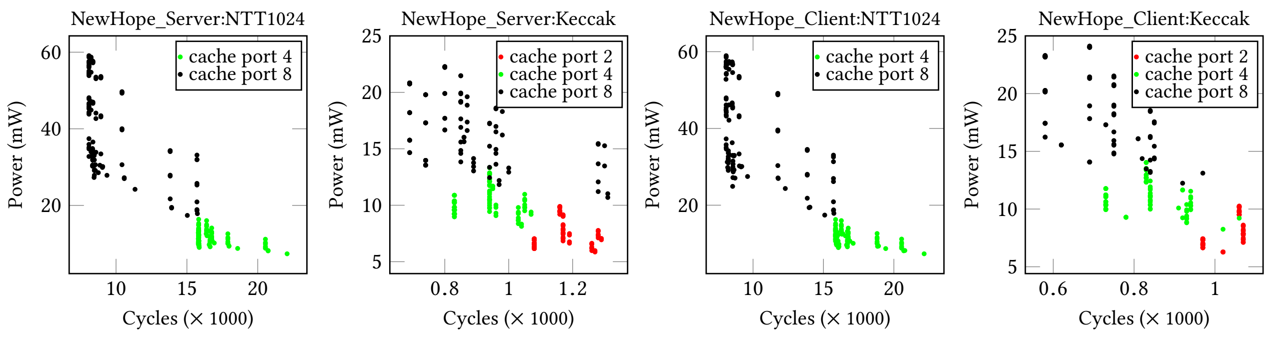
5 Evaluation

In this paper, we select two popular lattice-based cryptography schemes: Newhope and BLISS-BI. We explore the design space to build accelerators for the most time consuming kernels in each scheme with the primary goals of optimizing the overall performance of the system and satisfy programmability. The architectural template for the polynomial multiplication (`NTT1024` for NewHope and `FFT512` for BLISS-BI) and `Keccak-f[1600]` kernels can be seen in Figure 3. `Keccak-f[1600]` includes bit-wise arithmetic operation (shift, and, not, xor). Each lane of the `Keccak-f[1600]` accelerator has basic components to compute the F[1600] permutation function. Since `Keccak-f[1600]` has the same C-code implementation for both NewHope and BLISS-BI, we design a single accelerator with different parameters. In the case of `NTT1024` and `FFT512`, each datapath lane performs the butterfly operation; however, the logic (instructions) inside the `NTT1024` is different from `FFT512` since `NTT1024` performs the computation in the Montgomery [Mon85] domain. Consequently, two separate accelerators are designed for `NTT1024` and `FFT512`.

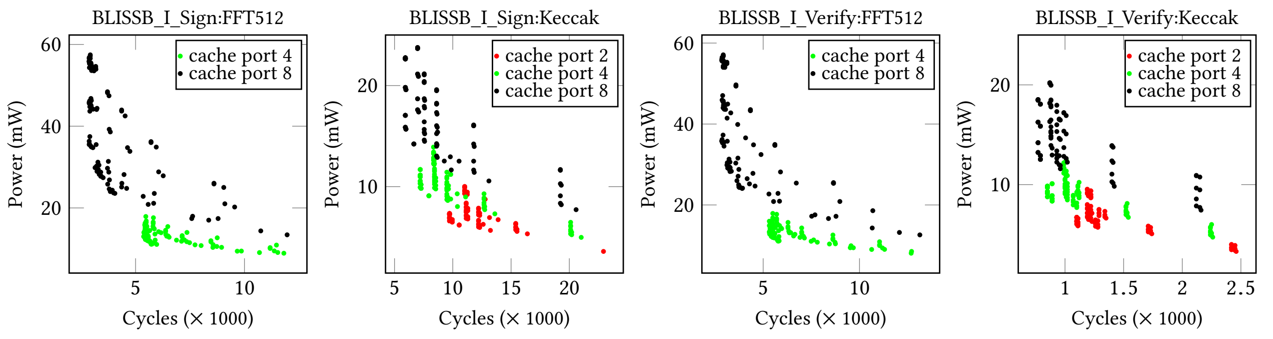
The optimized kernel of NTT1024 can be used inside any variant of NewHope that employs the same function as the polynomial multiplication with $n = 512, 1024$. Similarly, FFT512 can be used to accelerate other versions of BLISS-B with $n = 512$ or $n = 256$. However, `Keccak-f[1600]` is the SHA-3 NIST standardized function that can be used in any scheme to generate pseudo random numbers or securely hash a message.

Figure 4 shows the trade-off between latency and power of the accelerator for different kernels. We observe that some parts are more critical in the design space: long tail regions and sharp shaped regions. In the long tail regions, with almost the same level of the power, a wide range of the performance can be gained. In other words, inside the long tails, saving a negligible amount of power consumption can remarkably hurt performance and therefore we should select the fastest design point in that region. These long tail regions can be seen in NTT1024 (Figure 4a) and FFT512 (Figure 4b) between 50k to 100k cycles. In the sharp shape regions, a small change in one parameter (e.g., cycles) can lead to remarkable change in the other parameter (e.g., power); for instance, in order to improve the performance of NTT1024 by (1%), 3× more power should be consumed by increasing cache ports from 2 to 8.

NTT1024 and FFT512 have a wide design space (from 10k cycles to 220k cycles for NTT1024 and from 7k



(a) NewHope



(b) BLISS-BI

Figure 5: Sensitivity of the accelerators to the number of cache ports. To investigate sensitivity of the accelerator's power and performance to the number of cache ports, accelerators' cache port is set to 1 and all other parameters are swept; afterwards, design points that improve the overall performance of the system are extracted. We perform similar sensitivity analysis for cache port sizes of 2,4 and 8. For instance, all the green points are the designs with cache port size of 4.

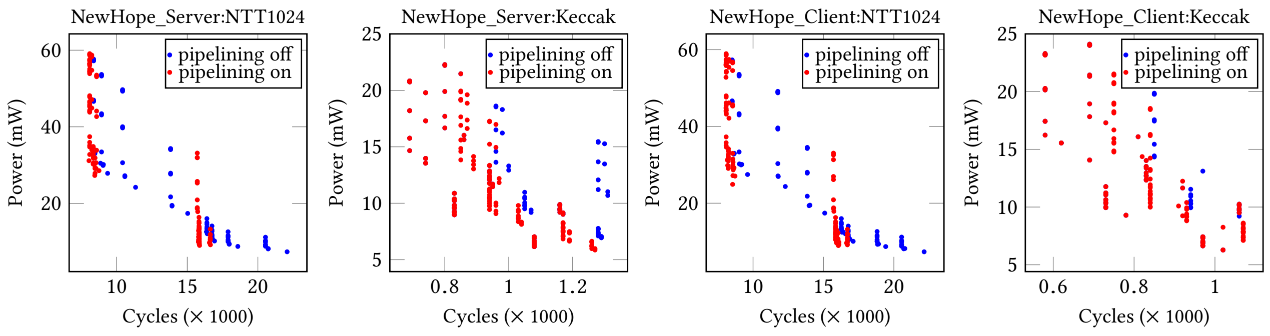
to 240k cycles for FFT512). Only a small range of the design space can improve the overall performance of the schemes; we only consider these beneficial points in Figure 5-8 to evaluate the sensitivity of the power and performance of the accelerators to each design parameter.

5.1 Design parameters

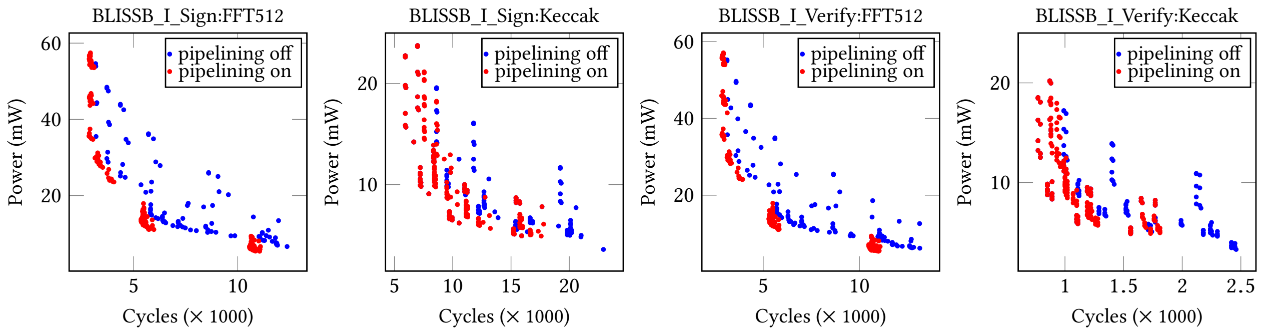
In this section, sensitivity of the latency and power of the accelerator to different design parameters (compiler and micro-architectural) are discussed for NewHope (client and server) and for BLISS-BI (sign and verify). Among all the sweepable parameters, we select those with remarkable impact on the latency and power of the accelerator.

Figure 5 presents the effect of number of cache ports on the power and latency of the accelerators for Newhope (5a) and BLISS-BI (5b). Figure 5a has four columns; the two left columns are the results for NewHope server (NewHope_Server:NTT1024 and NewHope_Server:Keccak) and two right columns are those of NewHope client (NewHope_Client:NTT1024 and NewHope_Client:Keccak).

Cache port. Increasing the number of cache ports considerably improves the latency of all accelerators with the cost of higher power consumption. The number of cache ports can improve the performance of the accelerator since it allows different lanes of the accelerator to perform memory operations in parallel. Based on Figure 5, assuming the performance as the main goal, the best cache port is 8 which reaches the highest speedup. Taking power into consideration, cache port 4 would be preferable since almost for all the kernels it yields decent performance and power consumption. For NewHope (both client and server) there is no design point with cache ports of 1 that can improve the overall performance of the system when either NTT1024 or



(a) NewHope



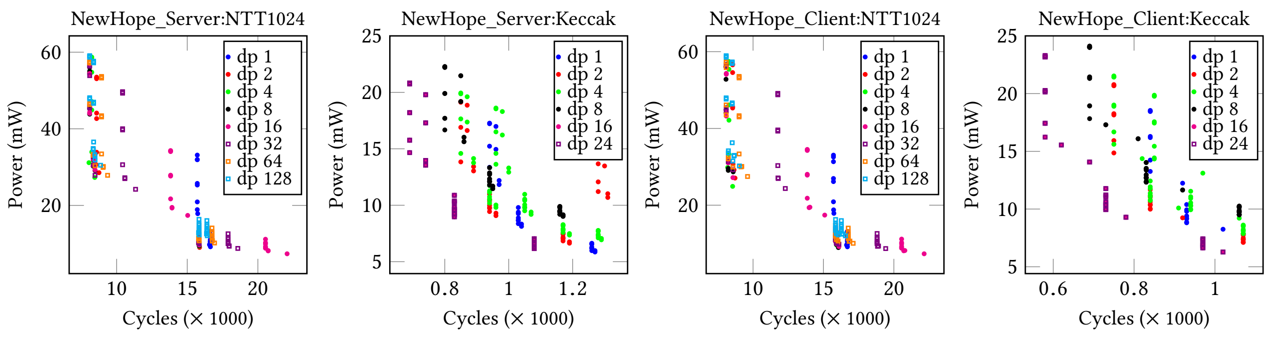
(b) BLISS-BI

Figure 6: Sensitivity of the accelerators to loop pipelining. Loop pipelining is disabled and all other design parameters are swept (blue points). Red points are all the design points with enabled loop pipelining.

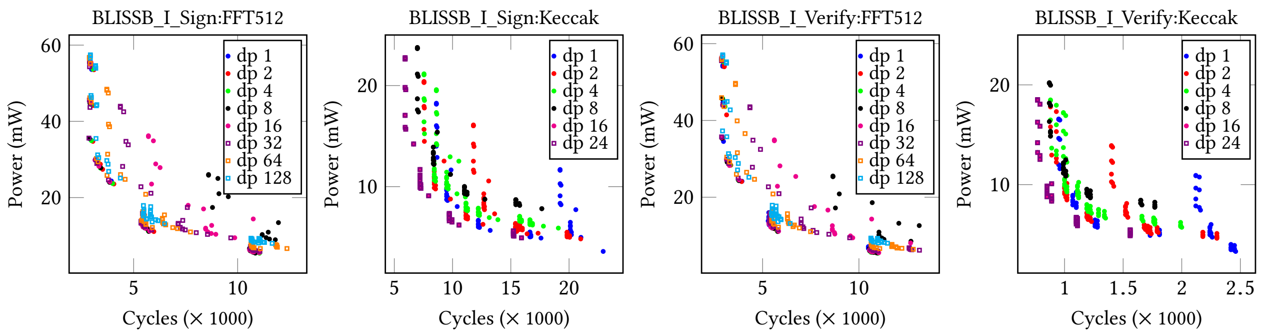
`Keccak-f[1600]` is accelerated. The same trend can be seen for BLISS-BI sign and verify.

Loop pipelining. Software (loop) pipeline is a compiler optimization technique which exploits the loop level parallelism to allow out-of-order execution of the instructions in different loop iterations. Due to the iterative nature of the cryptographic kernels, software pipelining can be beneficial to improve the performance: for instance, in `Keccak-f[1600]`, the result of one loop iteration is used in the next loop iteration. Consequently enabling software pipelining results in significant performance improvement at the cost of consuming more power. As can be seen in Figure 6, most of the blue points (pipelining off) are located in the slow power efficient areas. Software pipelining is closely coupled with another compiler level optimization, loop unrolling, as discussed next.

Loop unrolling. Loop unrolling is a compiler optimization technique that trades performance for resources (hardware component and the code size). Loop unrolling provides speedup by reducing the branch penalty along with hiding the memory latency. Besides, if different iterations of the loop are independent, they can be executed in parallel. In gem5-Aladdin, loop unrolling determines number of datapath lanes that are created inside the accelerator for each loop which is consistent with the high level synthesis tools. Each of the kernels in this study has only one loop hence unrolling factor equals to the number of datapath lanes in the accelerator. Performance and power of the accelerator are more sensitive to the number of parallel lanes, thus we sweep this parameter with finer granularity (Figure 7). For all the kernels, there is a saturation point where increasing number of the lanes does not have notable impact on the accelerator's performance. For NTT1024, an accelerator with one cache port and one lane offers the lowest power consumption by sacrificing the performance; doubling number of lanes improves the performance about twofold. With loop pipelining enabled, increasing the parallel datapath lanes from 2 to 128 yields 10% improvement in performance with almost the same power consumption. The unrolling factor of 4 is suggested since further increase in the number of lanes does not improve the performance with the cost of more energy consumption. For FFT512 (BLISS-BI), increasing the datapath lanes from 1 to 2 slightly shifts the Pareto curve to the left. Datapath lanes larger than 8 results in small changes in the latency and power at the cost of larger area. Considering the area, 8 parallel lanes reaches the best performance



(a) NewHope



(b) BLISS-BI

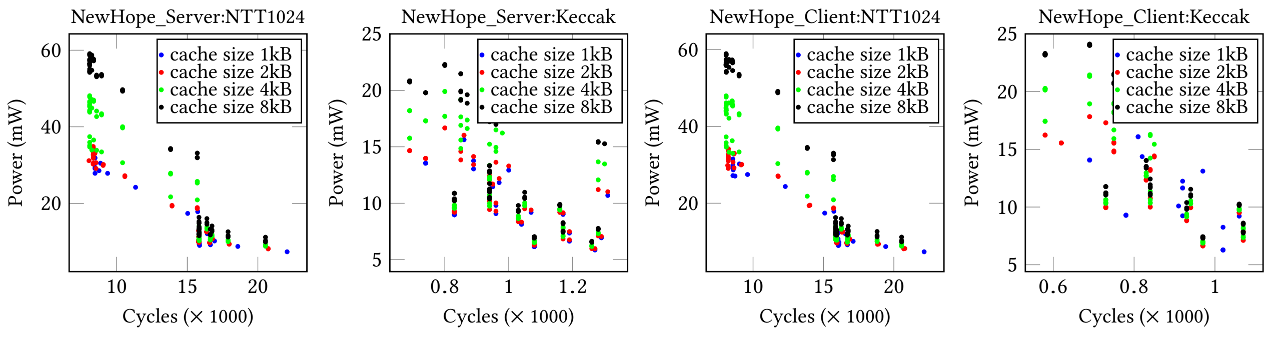
Figure 7: Sensitivity of the accelerators to the number of datapath lanes. Number of parallel lanes (loop unrolling factor) is fixed at 1 and all other design parameters are swept. This process is performed for valid parallel lanes in the range from 1 to 24 for `Keccak-f[1600]` and 1 to 128 for `NTT1024` and `FFT512`.

with the lowest EDP. For `Keccak-f[1600]` in both NewHope (client and server) BLISS-BI (sign and verify), increasing number of lanes from 1 to 2 slightly improves the performance; however, the number of lanes equals to 4 and 8 yields almost the same performance with higher power consumption. 16 parallel data path lanes (maximum parallel lanes that `Keccak-f[1600]` can have inside gem5-Aladdin) reaches the fastest designs by notably shifting the curve to the left.

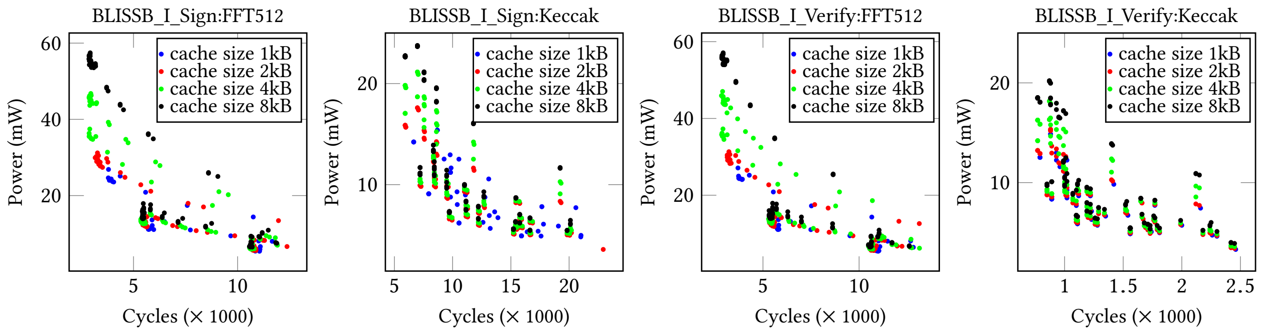
Cache size. As shown in Figure 8, we sweep cache size in the range of [1kB, 2kB, 4kB, 8kB]. Increasing the cache size of the accelerator improves the memory access by reducing the miss rate penalty due to the fine-granularity of the caches that leads to overlapping the cache misses with computation. NTT1024 is a compute intensive kernel, hence increasing the cache size of the accelerator has negligible performance improvement; however, larger cache sizes results in a moderate increase in the power consumption. NTT1024 accelerator with cache size of 2kB improves performance by 4% over 1kB cache size. Increasing the cache size from 2kB to 4kB and 8kB improve performance by less than 1% with 50% and 100% more power. Considering area and power, cache size of 2kB is suggested. Similarly cache size of 2kB is preferable for `Keccak-f[1600]` for both the server and the client in NewHope and signer and verifier in BLISS-BI. In the case of the FFT512 in BLISS-BI, cache size of 2kB is preferable for verifier that is running on a resource constrained device. For the signer, cache size of 4kB is suggested.

Cache associativity. This parameter has negligible effect on the performance of `Keccak-f[1600]` kernel in BLISS-BI and NewHope. For NTT1024 and FFT512, cache associativity of 2 slightly improves the performance. For all the kernels, cache associativity of 4 is suggested by taking power and area into account with almost the same performance as associativity of 2.

Configurable parameters. Cache line size and system bus width are two sweepable parameters that are shared over the SoC. Cache line size of the accelerator equals to the cache line size of the host CPU. Increasing the cache line size and system bus width can significantly improve the performance. However, we assume that

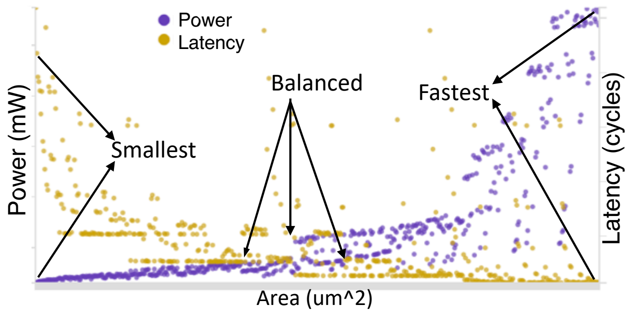


(a) NewHope



(b) BLISS-BI

Figure 8: Sensitivity of the accelerators to the cache size. Size of the cache is fixed at 1kB and all other design parameters are swept. This process is performed for valid cache sizes in the range from 1kB to 8kB.



(a) Visualization of different design points

Design points	Accelerator			Sysem (CPU + Accelerator)
	Latency (Cycles)	Power (mW)	Area (KEG)	Latency (Cycles)
Fastets	2854	35.60	567	1711686
Smallest	11053	5.4	189	1875907
Balanced	5707	11.181	311	1769885

(b) Numerical details of different design points

Figure 9: Elastic Search visualization and pruning (kernel is FFT512 in the sign step of BLISS-BI).

the host processor and the system bus is not configurable, hence we set the cache line size and system bus width to be 64-byte and 8-byte, respectively.

There are other design parameters that we set to be constant. Clock frequency of the accelerator, system bus and the host CPU are fixed at 100MHz, 1GHz and 2GHz respectively. Other constant parameters include cache line flush latency, cache line invalidate latency, accelerator TLB size, and TLB miss penalty.

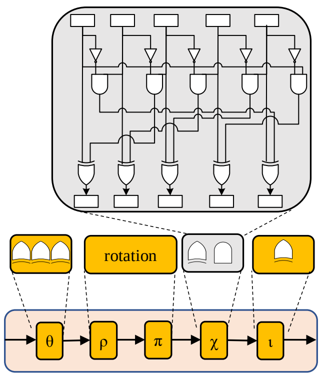
6 Selected Design Points

Figure 10a and 10b show the microarchitecture of a single lane of the `Keccak-f[1600]` and NTT (butterfly) accelerator from the architectural template (Figure 3). Power of the accelerator (tens of mW) is 3 orders-of-

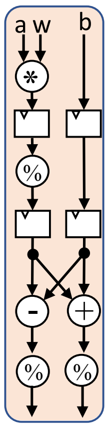
Table 1: Latency and EDP improvement of the schemes (functional units power models are based on OpenPDK 45nm and SRAM model from CACTI 5.3) when only one of the accelerators (at 100MHz) is attached to the host x86 CPU (at 2GHz).

Scheme	Accelerated kernel	Improvement (%)		Area (mm^2)	Aera (KGE[I])	32-bit logics				
		Latency	EDP			Adders	Bit-wise Operators	Multipliers	Registers	Shifters
NewHope Server	**NTT1024**	35	49.3	0.681	433	8	24	19	200	8
	Keccak	1.4	2.8	0.776	493	0	0	0	1074	0
NewHope Client	**NTT1024**	30.6	43.9	0.707	449	8	24	23	192	8
	Keccak	1.1	2.3	0.776	493	0	0	0	1074	0
BLISS-BI Sign	**FFT512**	8	12.5	1.144	727	160	224	32	1076	128
	Keccak	14.3	28.16	0.766	487	0	0	0	1074	0
BLISS-BI Verify	**FFT512**	42.1	60.6	1.377	875	160	224	32	1074	128
	Keccak	7.7	14.17	0.785	499	0	0	0	1074	0

[I] kilo gate equvalent



(a) Keccak-f[1600]



(b) Butterfly (for NTT1024 and FFT512)

Figure 10: Microarchitecture of a single datapath lane of the (a) `Keccak-f[1600]` and (b) butterfly accelerator.

magnitude lower than that of a X86 CPU (about 6W), hence it make sense to expedite finishing the accelerator job for resourceful systems. We choose performance to be accelerator's design goal and hence the design point with the lowest latency is selected to be attached to the CPU. Table 1 shows the improvement of the latency and EDP of the schemes when only one kernel is accelerated and rest of the scheme remains untouched.

BLISS-BI. Latency (cycles) and EDP of the verify step in are improved 42% and 60% when all the invocations to the FFT512 are offloaded to the accelerator and calls to the `Keccak-f[1600]` are performed on the host CPU. 8% and 12% improvement in the latency and EDP of the sign step scheme is achieved when only FFT512 kernel is accelerated. Accelerating the `Keccak-f[1600]` kernel in sign step scheme results in 14% and 28% enhancement of latency and EDP. Suppose $time_{kernel}$ is the number of CPU cycles (2GHz) that CPU spends inside a kernel without the accelerator. Let $time_{accelerator}$ be the CPU cycles (2GHz) from the time the CPU invokes the accelerator until the completion of the acceleration task. $time_{accelerator}$ is 3.1$\times$ and 2.6$\times$ smaller than $time_{kernel}$ for FFT512 and `Keccak-f[1600]`, respectively.

NewHope. In the case of NewHope server, the latency (EDP) of the scheme is improved by 35% (49%) and 1.4% (2.8%), respectively, when NTT1024 and `Keccak-f[1600]` accelerators are attached to the CPU. Accelerating the NTT1024 in client side of NewHope improves performance and EDP by 30% and 43%. $time_{accelerator}$ is 2.1$\times$ and 1.4$\times$ smaller than $time_{kernel}$ for NTT1024 and `Keccak-f[1600]`, respectively.

6.1 Combined Design Points

After selecting the fastest design points, two accelerators with different architectural characteristics are attached to the CPU and communicate via the coherent memory hierarchy (Figure 3). Frequency of the CPU is 2GHz, while the accelerators have a clock frequency of 100MHz. Table 4 shows the results for the latency and EDP of

Table 2: Delay and EDP improvement of the schemes when both accelerators (fastest design points) are attached.

Scheme	**Improvement (%)**	
	Latency	**EDP**
NewHope Server	37	52
NewHope Client	33	48
BLISS-BI Sign	24	44
BLISS-BI Verify	47	67

Table 3: Performance improvement of BLISS-BI in terms of number of generated and verified signatures in one second. Benchmarking is performed with gem5 (X86 at 2GHz and 4GB RAM).

Scheme	**Speedup**	
BLISS-BI	**Sign/s**	**Verify/s**
	× 1.3	× 1.9

NewHope and BLISS-BI when two accelerators are attached to the CPU. Latency and EDP of the sign step of BLISS-BI improved by 24% and 44%, respectively. These improvements are 47% and 67% for the verify step.

6.2 Design For Resource Constrained Systems

Table 4 demonstrates the trade-off between area and latency of the accelerator for BLISS-BI scheme. The best design points in Section 6 are the fastest ones. A slight decrease in the overall performance of the system (host CPU and the accelerator) results in considerable saving in the area and energy of the accelerator. Normalized to the fastest design point, by only 1.6% increase in the system's latency (cycles) when the *smallest* `Keccak-f[1600]` accelerator is attached to the verifier's host CPU, 33.9% and 67.3% saving in the energy and area of the accelerator can be achieved. A middle ground `Keccak-f[1600]` accelerator for verifier can save 48% of the area with with only 0.5% increase in the latency. These types of exploration results demonstrate the utility and efficacy of the DSE framework for exploring a range of implementation options tuned to the needs of specific deployment scenarios.

7 Prior Art

Digital accelerators can be divided into programmable (e.g., GPUs and DSPs), dynamic datapath (e.g., Dyser [GHN+12]) and fixed-function accelerators (e.g., C-Cores [VSG+10]). Accelerators have been widely employed in various domains such as deep neural networks [SCYE17], image encryption algorithms [ASN15], classical (e.g., AES [ANM13] and AES-NI [Gue10]) and post-quantum cryptography. Plenty of post quantum cryptographic software and hardware accelerators have been proposed [NDR+17]. Until now, FPGAs are the only hardware

Table 4: Area and performance trade-off of the accelerator; baseline is the fastest design point.

Scheme	**Kernel**	**Design point**	**Increase (%)**		**Improvement in acceletors(%)**		**Area (KEG)**
			Accel's latency	**System's latency**	**Accel's energy**	**Accel's area**	
BLISS-BI Sign	**FFT512**	**smallest**	287	9.5	41	74	189
		balanced	99	3.4	37.2	57.2	311
	Keccak	**smallest**	76	4.8	30	67	161
		balanced	21.6	1.3	23.7	44.6	270
BLISS-BI Verify	**FFT512**	**smallest**	286	56	54	78	192
		balanced	11	2.2	29.6	34.9	569
	Keccak	**smallest**	45.6	1.6	33.9	67.3	163
		balanced	15.7	0.5	26.9	48	259

target platform of accelerators for post quantum cryptographic schemes, such as key exchange mechanism [TT17], public key encryption [PG13], digital signature [PDG14] and homomorphic encryption [PNPM15]. FPGAs provide flexibility and reconfigurability; however the proposed FPGA-based accelerators lack agility.

Fast exploration of the design space for accelerators is important for academia and industry to reduce the time-to-market. To the best of our knowledge, there is no prior fast exploration of the design space of programmable accelerators for post quantum cryptographic schemes.

8 Limitations

Although our proposed DSE framework can expediently perform the DSE for a specific domain, we currently lack the ability to verify estimations of area and power generated by the system, and rely on prior art for the accuracy of the gem5-Aladdin simulator. Our next step in this work is to verify the area, power and of the accelerator with actual hardware implementations.

9 Conclusion

In this paper, we present a rapid design flow that enables exploration, comparative analyses, and the design of a family of domain specific accelerators for common cryptographic schemes based on ideal lattices. This design flow enables agility in the face of rapidly changing protocols and standards, and also allows for customization of accelerators to span the gamut from resource-and-energy constrained IoT devices, to high-performance server-class compute platforms. For example, the accelerator designed for NTT1024 can be leveraged by any variant of the scheme that uses Gentleman-Sande butterfly construction [GS66], either with $n = 512$ or $n = 1024$ to reach $2\times$ improvement in the polynomial multiplication. We use the implementation of polynomial multiplication from [Saa17] to design accelerators for FFT512, which can be leveraged by BLISS-B variants (BLISS-BII, BLISS-BIII, and BLISS-BIV) and BLZZRD [Saa17] for the digital signature scheme. Finally domain specific accelerators for `Keccak-f[1600]` can be leveraged in any cryptographic scheme, implementing either lattice based cryptography or classic cryptography. Future work will address hardware validation of our early DSE results, as well as the development of a library of templatized programmable accelerators for emerging post-quantum cryptographic schemes.

References

[ADPS15] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Annual ACM Symposium on Theory of Computing*, STOC '96, 1996.

[ANM13] Ali Ansarmohammadi, Hamid Nejatollahi, and Ghasemi Mehdi. A low-cost implementation of aes accelerator using hw/sw co-design technique. In *CADS*, 2013.

[ASN15] Ali Ansarmohammadi, Saeed Shahinfar, and Hamid Nejatollahi. Fast and area efficient implementation for chaotic image encryption algorithms. In *2015 18th CSI International Symposium on Computer Architecture and Digital Systems*, CADS '15, 2015.

[BBB$^+$11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 2011.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. 2013.

[Ber08] Daniel J Bernstein. Chacha, a variant of salsa20. In *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*, 2008.

[BG14] Shi Bai and Steven D Galbraith. An improved compression technique for signatures based on learning with errors. In *Proceedings of topics in Cryptology*, CT-RSA '14, 2014.

[BHS18] Song Bian, Masayuki Hiromoto, and Takashi Sato. Dwe: Decrypting learning with errors with errors. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, 2018.

[Bra16] Matt Braithwaite. Experimenting with post-quantum cryptography, 2016.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '13. 2013.

[Duc14] Léo Ducas. Accelerating bliss: The geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874, 2014.

[Dwo15] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[GHN$^+$12] Venkatraman Govindaraju, Chen-Han Ho, Tony Nowatzki, Jatin Chhugani, Nadathur Satish, Karthikeyan Sankaralingam, and Changkyu Kim. Dyser: Unifying functionality and parallelism specialization for energy-efficient computing. In *Proceedings of the 2012 45rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '12, 2012.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*, CHES '12, 2012.

[GMRS$^+$15] Oscar Garcia-Morchon, Ronald Rietman, Sahil Sharma, Ludo Tolhuizen, and Jose Luis Torre-Arce. Dtls-himmo: Achieving dtls certificate security with symmetric key overhead. In *Proceedings of the European Symposium on Computer Security*, ESORICS '15, 2015.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, STOC '08, 2008.

[GS66] W. M. Gentleman and G. Sande. Fast fourier transforms: For fun and profit. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, AFIPS '66, 1966.

[Gue10] Shay Gueron. Intel® advanced encryption standard (aes) new instructions set. *Intel Corporation*, 2010.

[Lon01] Gui-Lu Long. Grover algorithm with zero theoretical failure rate. *Physical Review A*, 2001.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Proceedings of the Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, 2010.

[LS12] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive, Report 2012/090, 2012.

[Lyu09] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, 2009.

[Mon85] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 1985.

[MR09] Daniele Micciancio and Oded Regev. *Lattice-based cryptography.* 2009.

[NDC17] Hamid Nejatollahi, Nikil Dutt, and Rosario Cammarota. Trends, challenges and needs for lattice-based cryptography implementations: Special session. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion*, CODES '17, 2017.

[NDR+17] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. Software and hardware implementation of lattice-cased cryptography schemes. *University of California Irvine, CECS TR 17-04*, 2017.

[Nus80] Henri Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1980.

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*, CHES '14, 2014.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In *Proceedings of the International Workshop on Post-Quantum Cryptography*, PQCrypto '14, 2014.

[PG13] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *Proceedings of the Revised Selected Papers on Selected Areas in Cryptography*, SAC '13, 2013.

[PNPM15] Thomas Pöppelmann, Michael Naehrig, Andrew Putnam, and Adrian Macias. Accelerating homomorphic evaluation on reconfigurable hardware. In *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*, CHES '13, 2015.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. 2005.

[Saa17] Markku-Juhani O Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 2017.

[SCYE17] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 2017.

[Sha16] Vishal Sharma. *Getting Started with Kibana.* 2016.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 1997.

[SRWB14] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, 2014.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, 2009.

[SXS+16] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Y. Wei, and D. Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '16, 2016.

[TT17] Oder Tobias and Güneysu Tim. Implementing the newhope-simple key exchange on low-cost fpgas. In *Proceedings of the International Conference on Cryptology and Information Security in Latin America*, LATINCRYPT '17, 2017.

[VSG+10] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '10, 2010.

[WJ96] S. J. E. Wilton and N. P. Jouppi. Cacti: an enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 1996.

[WWA01] L. Wu, C. Weaver, and T. Austin. Cryptomaniac: a fast flexible architecture for secure communication. In *Proceedings 28th Annual International Symposium on Computer Architecture*, 2001.