No Silver Bullet: Optimized Montgomery Multiplication on Various 64-bit ARM Platforms

1st Hwajeong Seo
College of IT Engineering,
Hansung University,
Seoul, Republic of Korea,
hwajeong84@gmail.com

2nd Pakize Sanal
Department of Computer, Electrical Engineering and Computer Science,
Florida Atlantic University,
Boca Raton, USA
psanal2018@fau.edu

3rd Wai-Kong Lee
Department of Computer Science,
Gachon University,
Seongnam, South Korea,
waikonglee@gachon.ac.kr

4th Reza Azarderakhsh
Department of Computer, Electrical Engineering and Computer Science
Florida Atlantic University,
Boca Raton, USA
razarderakhsh@fau.edu

***Abstract*—In this paper, we firstly presented optimized implementations of Montgomery multiplication on 64-bit ARM processors by taking advantages of Karatsuba algorithm and efficient multiplication instruction sets for ARM64 architectures. The implementation of Montgomery multiplication improved the performance of public key cryptography (e.g. CSIDH, ECC, and RSA) implementations on ARM64 architectures, directly. Last but not least, the performance of Karatsuba algorithm does not ensure the fastest speed record, while it is determined by the clock cycles per multiplication instruction of target ARM architectures. In particular, recent Apple processors based on ARM64 architecture show lower cycles per instruction of multiplication than that of ARM Cortex-A series. For this reason, the schoolbook method shows much better performance than the sophisticated Karatsuba algorithm on Apple processors. With this observation, we can determine the proper approach for multiplication of cryptography library (e.g. MS-SIDH) on Apple processors and ARM Cortex-A processors.**

***Index Terms*—Montgomery Multiplication, ARM64, Public Key Cryptography, Software Implementation**

I. Introduction

The modular reduction is the fundamental building block of conventional public key cryptography (e.g. RSA [1], El-Gamal [2], and ECC [3], [4]) to post-quantum cryptography (e.g. RLWE [5], SIDH [6], and CSIDH [7]). One of the most well-known modular reduction techniques is Montgomery algorithm [8]. This approach replaces the complicated division operation for the modular reduction in relatively simple multiplication operations. For this reason, efficient implementations of Montgomery multiplication on target processors have been actively studied. In this paper, we firstly introduce the optimized Montgomery multiplication on ARM64 processors and show the impact on public key cryptography protocols (i.e. CSIDH)[1]. Furthermore, we found that recent Apple processors provide the multiplication with very low latency. This nice feature leads to the new direction to implement the multiplication on Apple processors (i.e. simple schoolbook approach rather than sophisticated Karatsuba algorithm). With this observation, we can improve the performance of cryptography libraries based on Karatsuba algorithm (e.g. MS-SIDH) on recent Apple processors by replacing the multiplication to the schoolbook method[2].

The remainder of the paper is structured as follows: We review the related work on Montgomery multiplication, Karatsuba algorithm, and ARM64 processors in Section II. We present the optimized implementation of Montgomery multiplication on ARM64 processors in Section III. In Section IV, we present results on various 64-bit ARM platforms. We end with conclusions in Section V.

II. Related Works

A. Montgomery Multiplication

Montgomery multiplication consists of multiplication and reduction parts. The multiplication can be implemented in different ways by altering the order of operands and intermediate results. The operand-scanning method performs a multiplication in a row-wise manner. This approach is suitable for processors with many registers to retain long intermediate results. The alternative approach is the Comba (i.e. product-scanning) method [9]. Partial products are computed in a column-wise manner and only small number of registers is required to maintain the intermediate result. Furthermore, since all partial products of each word of the result are computed and added consecutively, the final result word is obtained directly and no intermediate results have to be stored or loaded in the algorithm In [10], a hybrid-scanning method combining two aforementioned methods is presented. Afterward, several optimized implementations for multiplication were suggested by caching operands [11], [12]. However, the complexity of partial products for N-word multiplication is N^2.

Karatsuba algorithm computes a multiplication with only three partial products compared to four that are required by

[1]The proposed method is applicable to RSA, and ECC, as well.

[2]https://github.com/microsoft/PQCrypto-SIDH/tree/master/src

Algorithm 1 Montgomery Reduction

```
Require: An $m$-bit modulus $M$, Montgomery radix $R = 2^m$,
    two $m$-bit operands $A$ and $B$, and $m$-bit pre-computed
    constant $M' = -M^{-1} \bmod R$
Ensure: Montgomery product ($Z = (A \cdot B) \cdot R^{-1} \bmod M$)
1: $T \leftarrow A \cdot B$
2: $Q \leftarrow T \cdot M' \bmod R$
3: $Z \leftarrow (T + Q \cdot M)/R$
4: if $Z \geq M$ then $Z \leftarrow Z - M$ end if
5: return Z
```

aforementioned multiplication methods [13]. The number of partial products is estimated by $N^{log_2 3}$, which is a great improvement compared to N^2 of the standard multiplication. The previous multiplication on ARM64 processors mainly utilized the Karatsuba algorithm for optimal performance.

Montgomery algorithm was firstly proposed in 1985 [8]. Montgomery algorithm avoids the division in modular multiplication by introducing simple shift operations. Given two integers A and B and the modulus M, to compute the product $P = A \cdot B \bmod M$ in Montgomery method, operands A and B are firstly converted into Montgomery domain (i.e. $A' = A \cdot R \bmod M$ and $B' = B \cdot R \bmod M$). For efficient computations, Montgomery residue R is selected as a power of 2 and constant $M' = -M^{-1} \bmod 2^n$ is pre-computed. To compute the product, following three steps are conducted: (1) compute $T = A \cdot B$; (2) perform $Q = T \cdot M' \bmod 2^n$; (3) calculate $Z = \frac{(T+Q\cdot M)}{2^n}$ and (4) compute final reduction $Z \leftarrow Z - M$ if $Z \geq M$. Detailed descriptions of Montgomery reduction is available in Algorithm 1.

There are two approaches, including separated and interleaved ways, for Montgomery multiplication. The interleaved approach reduces the number of memory access for intermediate result, but many implementations on ARM64 selected the separated approach. The separated implementation can employ the most optimal approach for multiplication and reduction operations each. In this paper, we also selected the separated approach. Both multiplication and Montgomery reduction parts are optimized with Karatsuba algorithm. In particular, we utilized the Karatsuba based Montgomery reduction by [14].

In Table I, the comparison of Montgomery multiplication on 64-bit ARMv8 Cortex-A processors is given. Previous works [15]–[17] target for Montgomery friendly prime and it's application is limited to only SIKE due to the special prime form. On the other hand, the proposed Montgomery multiplication is targeting for random prime and it's application is CSIDH, RSA, and ECC.

B. 64-bit ARMv8 Processors

ARMv8 is a 64-bit architecture for high-performance embedded applications. The 64-bit ARMv8 processors support both 32-bits (AArch32) and 64-bits (AArch64) architectures. It provides 31 general purpose registers which can hold 32-bit values in registers `w0-w30` or 64-bit values in registers `x0-x30`. ARMv8 processors started to dominate the smartphone market soon after the release in 2011 and nowadays they are widely used in various smart phones (e.g. iPhone and Samsung Galaxy series) and laptop (e.g. MacBook Pro). Since the processor is used primarily in embedded systems, smart phones and laptop computers, efficient and compact implementations are of special interest. ARMv8 provides two 64-bit multiplication instructions, `MUL` and `UMULH`, both of which carry out one half of a 64×64-bit multiplication. In both cases, inputs are 64-bit registers. `MUL` computes the lower 64-bit half of results while `UMULH` computes the higher 64-bit half. Details of ARMv8-A architecture can be found in [19].

Implementation	Multiplication	Montgomery Reduction	Application
Liu et al. [15]	Karatsuba	–	–
Seo et al. [16]	Karatsuba	Product Scanning	SIKE
Seo et al. [17]	Karatsuba	Product Scanning	SIKE
Jalali et al. [18]	Operand Scanning	Operand Scanning	CSIDH, RSA, ECC
This work	Karatsuba	Karatsuba	CSIDH, RSA, ECC

TABLE I
COMPARISON OF MONTGOMERY MULTIPLICATION ON 64-BIT ARMV8 CORTEX-A PROCESSORS.

III. PROPOSED IMPLEMENTATIONS

A. Optimization of Montgomery multiplication

One of the most expensive operation for public key cryptography is modular multiplication. In this paper, we present the optimal modular multiplication implementation in the separated way for 64-bit ARM architectures.

First, the multi-precision multiplication is performed in Karatsuba algorithm. As described in [16], 2-level Karatsuba computations are performed for 512-bit multiplication on 64-bit ARM architectures. We further optimized the memory access by using general purpose registers to retain operands. In particular, Karatsuba multiplication needs to update operands but these operands are used in following computations. In this case, we keep operands in registers to avoid frequent memory loading operations.

Multiplication and reduction operations are implemented in one function to avoid the function call and register `push`/`pop` instructions. Intermediate results of multiplication are stored in `STACK` memory and the result is directly used in the modular reduction.

For the computation of Montgomery reduction, the product ($Q \leftarrow T \cdot M' \bmod R$) is performed (Step 2 of Algorithm 1) in ordinary way. Afterward, the product $Q \cdot M$ is computed in a hybrid way (Step 3) [14]. The complexity of N-word original Montgomery reduction is $N^2 + N$ word-wise multiplications, while the hybrid Montgomery reduction is $\frac{7N^2}{8} + N$ word-wise multiplications. In particular, the hybrid Montgomery reduction consists of two 256-bit Montgomery reduction in product-scanning approach and two 256-bit (1-level) Karatsuba multiplication operations.

28 out of 31 registers are utilized for 512-bit Montgomery reduction on the ARM64 architecture. The detailed register utilization is given in Table II. When we perform the hybrid Montgomery reduction ($\frac{T+Q\cdot M}{2^n}$), computations ($Q_L \cdot M_L$ and $Q_H \cdot M_L$) are performed in sub-Montgomery reduction and

Modulus M	Quotient Q	Temporal registers	Constant M'
8	4	15	1

TABLE II
REGISTER UTILIZATION FOR MONTGOMERY REDUCTION ON ARM64.

Algorithm 2 Optimized implementation of 256-bit sub-Montgomery reduction on ARM64 processors.

```
Input: Modulus (M0-M3), intermediate
    results (C0-C3), constant (M_INV),
    temporal registers (T0-T3).
Output: Intermediate results (C0-C3),
    quotient (Q0-Q3).
 1: mul Q0, C0, M_INV

 2: mul T0, Q0, M0
 3: umulh T1, Q0, M0
 4: mul T2, Q0, M1
 5: umulh T3, Q0, M1

 6: adds C0, C0, T0
 7: adcs C1, C1, T1
 8: adcs C2, C2, xzr
 9: adcs C3, C3, xzr
10: adc C0, xzr, xzr
11: mul Q1, C1, M_INV
12: adds C1, C1, T2
13: adcs C2, C2, T3
14: adcs C3, C3, xzr
15: adc C0, C0, xzr

16: ...

17: mul T2, Q3, M3
18: umulh T3, Q3, M3
19: adds C1, C1, T0
20: adcs C2, C2, T1
21: adc C3, C3, xzr
22: adds C2, C2, T2
23: adcs C3, C3, T3
```

others ($Q_L \cdot M_H$ and $Q_H \cdot M_H$) are performed in Karatsuba multiplication, where L and H represent lower and higher parts of operand.

In Algorithm 2, the optimized implementation of 256-bit sub-Montgomery reduction on ARM64 processors is given. Partial products of reduction are performed in the product-scanning way. Three ARM64 instructions (MUL, UMULH, and ADD) are mainly utilized for partial products. Since multiplication operations require 6 clock cycles in Cortex-A series, the utilization of result directly incurs pipeline stalls [16]. In Line 1 of Algorithm 2, the quotient (`Q0`) is generated with 64-bit wise. This is simply performed with single `mul` instruction. In Line 2∼5, the quotient (`Q0`) is directly utilized, which incurs pipeline stalls. In Line 6∼7, the result of multiplication (`T0` and `T1`), which is computed in Line 2∼3, is accumulated to the intermediate result. This approach avoids the read-write dependency. In Line 10, the register is initialized with `xzr` instruction and the carry is obtained in `C0` register. In Line 11, the quotient (`Q1`) is generated but it is not utilized directly. In particular, the accumulation step (Line 12∼15) is performed with partial products in previous steps. This does not incur the read-write dependency. Following computations (after Line 16 to end) are performed in the similar way (i.e. read-write dependency free).

After the sub-Montgomery reduction, the remaining part is performed with the Karatsuba algorithm [16]. The additive Karatsuba algorithm performs the addition on the operand. This updates operands which cannot be used again. In the proposed implementation, we cached the operand in registers and this avoids the memory access for the operand re-loading. Afterward, one sub-Montgomery reduction and one Karatsuba multiplication are performed. Lastly, the final reduction is performed in the masked way. Firstly the intermediate result is subtracted by the modulus. When the borrow bit is captured, it sets the masked modulus. Otherwise, the modulus is set to zero. The result is subtracted by the masked modulus.

B. Acceleration of Public Key Cryptography

The proposed implementation of Montgomery multiplication is efficiently optimized. We can directly apply the proposed Montgomery multiplication to the CSIDH library by [18]. We checked the improved CSIDH implementation passed the CSIDH tests and pulic-key validation. Furthermore, conventional public key cryptography based on random prime (RSA and ECC) can also take advantages of proposed method.

IV. EVALUATION

The proposed implementation is evaluated on the various ARM64 architectures, which is largely divided into ARM Cortex-A and Apple A series. Detailed specifications for each processor is given in Table III.

In Table IV, the comparison of clock cycles for 512-bit Montgomery multiplication and constant-time CSIDH-P511 on 64-bit ARM architectures is given. The proposed implementations of 512-bit modular multiplication achieved performance enhancements than school-book method [18] by 1.25× and 1.23× on Odroid-C2 and Raspberry-pi4, respectively. Since the approach is generic, we can apply the proposed method to larger operand sizes. Montgomery multiplication is the fundamental operation in PKC. For the case study, we ported the implementation of Montgomery multiplication to CSIDH implementation. The performance of key exchange is improved by 1.16× and 1.23× than previous works on Odroid-C2 and Raspberry-pi4, respectively.

On the other hand, proposed implementations on Apple platforms show the opposite performance result. The schoolbook method based 512-bit modular multiplication achieved performance enhancements than proposed method by 0.71×, 0.65× and 0.69× on iPad mini5, iPhone SE2, and iPhone12 mini, respectively. The performance of key exchange is degraded by 0.71×, 0.64×, and 0.72× than previous works on Odroid-C2 and Raspberry-pi4, respectively.

In Table V, the comparison of cycles per instruction on ARM64 is given. The timing is measured with the average cycles after performing 1,000 times of each iteration without read-write dependency. The timing also includes function call and push/pop instructions. In ARM Cortex-A series, ratios of $\frac{MUL}{ADD}$ and $\frac{UMULH}{ADD}$ are 2.62∼4.34 and 4.65∼6.20 for Odroid-C2 and Raspberry-pi4 boards, respectively. This shows that the multiplication operation is more expensive than the addition operation on the target ARM Cortex-A architecture. For this reason, Karatsuba algorithm is effective on ARM Cortex-A series. On the other hand, ratios of $\frac{MUL}{ADD}$ and $\frac{UMULH}{ADD}$ on Apple A series are 1.34, 1.31, and 1.47∼1.38 for iPad mini5, iPhone SE2, iPhone12 mini, respectively.

In Table VI, the number of instructions for Montgomery multiplication methods is given. Schoolbook method (i.e. operand-scanning) requires 297 addition, 265 multiplication, and 134 memory access instructions, respectively. Compared with the Karatsuba algorithm, 225 addition instructions are

Company	Platform	Core	OS	Released	Revision	Decode	Pipeline depth	Out-of-order execution	Branch prediction	Exec. ports
ARM	Odroid-C2	Cortex-A53(@1.5GHz)	Ubuntu 16.04	2014	ARMv8.0-A	2-wide	8	×	Conditional	2
	Raspberry-pi4	Cortex-A72(@1.5GHz)	Ubuntu 20.10	2015	ARMv8.0-A	3-wide	15	○	○	8
Apple	iPad mini5	A12 (Vortex@2.49GHz)	iPadOS 14.4	2018	ARMv8.3-A	7-wide	16	○	-	13
	iPhone SE2	A13 (Lightning@2.65GHz)	iOS 14.4	2019	ARMv8.4-A	8-wide	16	○	-	13
	iPhone12 mini	A14(Firestorm@3.10GHz)	iOS 14.4	2020	ARMv8.4-A	8-wide	-	-	-	-

TABLE III

Comparison of ARMv8-A cores on ARM Cortex-A and Apple A processors.

Implementation	Odroid-C2 Timing [cc×10^6]			Raspberry-pi4 Timing [cc×10^6]			iPad mini5 Timing [cc×10^6]			iPhone SE2 Timing [cc×10^6]			iPhone12 mini Timing [cc×10^6]		
	[18]	Opt	[18]/Opt	[18]	Opt	[18]/Opt	[18]	Opt	[18]/Opt	[18]	Opt	[18]/Opt	[18]	Opt	[18]/Opt
Montgomery multiplication	1,309 cc	1,044 cc	1.25	973 cc	792 cc	1.23	167 cc	233 cc	0.71	154 cc	235 cc	0.65	150 cc	214 cc	0.69
Alice key generation	14,374	12,392	1.16	11,892	9,864	1.21	1,210	1,694	0.71	1,086	1,692	0.64	1,131	1,548	0.73
Bob key generation	14,386	12,392	1.16	12,098	9,916	1.22	1,212	1,681	0.72	1,086	1,693	0.64	1,132	1,557	0.73
Validation of Bob's key	58	50	1.16	43	35	1.21	8	11	0.71	7	11	0.65	7	10	0.72
Validation of Alice's key	58	50	1.16	43	35	1.21	8	11	0.71	7	11	0.66	7	10	0.72
Alice shared key generation	14,252	12,628	1.13	11,570	10,099	1.15	1,216	1,705	0.71	1,090	1,690	0.64	1,127	1,572	0.72
Bob shared key generation	14,544	12,555	1.16	12,453	10,114	1.23	1,214	1,681	0.72	1,084	1,706	0.64	1,135	1,546	0.73
Alice total computations	28,684	25,070	1.14	23,504	19,998	1.18	2,434	3,410	0.71	2,183	3,393	0.64	2,265	3,130	0.72
Bob total computations	28,988	24,998	1.16	24,594	20,065	1.23	2,433	3,373	0.72	2,177	3,409	0.64	2,274	3,113	0.73

TABLE IV

Comparison of clock cycles (×10^6) for Montgomery multiplication (for 512-bit) and (constant-time) CSIDH-P511 on 64-bit Odroid-C2, Raspberry-pi4, iPad mini5, iPhone SE2, and iPhone12 mini.

Platform	`MUL`	`UMULH`	`ADD`	`MUL/ADD`	`UMULH/ADD`
Odroid-C2	2.37	3.93	0.90	2.62	4.34
Raspberry-pi4	3.02	4.03	0.64	4.65	6.20
iPad mini5	0.57	0.57	0.42	1.34	1.34
iPhone SE2	0.49	0.49	0.37	1.31	1.31
iPhone12 mini	0.55	0.51	0.37	1.47	1.38

TABLE V

Comparison of cycles per instruction on ARM64.

Implementation	`ADD/SUB`	`MUL/UMULH`	`LDR/STR`
Schoolbook Method [18]	297	265	134
Karatsuba Algorithm	522	214	70

TABLE VI

Number of instructions for 512-bit Montgomery multiplication methods.

optimized away but it requires 51 multiplication and 64 memory access instructions, more than the Karatsuba approach. Due to the efficient multiplication instruction on Apple processors, reducing the number of multiplication by sacrificing the addition operation is not effective. For this reason, even though processors based on ARM64 architecture, the implementation technique should be different depending on cycles per multiplication instruction. This observation is useful for optimization of public key cryptography on ARM64. For example, Montgomery multiplication of MS-SIDH library is based on Karatsuba algorithm. This library can be improved by using operand-scanning method on Apple products.

V. Conclusion

In this paper, we presented optimized Montgomery multiplication implementations for the 64-bit ARM Cortex-A processors. Proposed implementations utilized the Karatsuba algorithm and ARMv8-A specific instruction sets. This work shows that proposed implementations on ARM Cortex-A platforms are more efficient than previous works. However, the platform with low multiplication latency (e.g. Apple A processors) achieved the better performance with the schoolbook method. This is because the evaluation of previous works is usually conducted on ARM Cortex-A processors. With the observation on this paper, the implementation should be evaluated on various ARM platforms for fair comparison and practicality.

The obvious future work is improving the MS-SIDH library on Apple A processors by utilizing the schoolbook method or other approaches (i.e. product-scanning and hybrid-scanning). Furthermore, we will investigate the multiplication method for various integer length. Lastly, proposed implementation will be the public domain and other cryptography engineers can directly use them for their cryptography applications.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[3] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*. Springer, 1985, pp. 417–426.

[4] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[5] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[6] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," in *Annual International Cryptology Conference*. Springer, 2016, pp. 572–601.

[7] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: an efficient post-quantum commutative group action," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 395–427.

[8] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[9] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM systems journal*, vol. 29, no. 4, pp. 526–538, 1990.

[10] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 119–132.

[11] M. Hutter and E. Wenger, "Fast multi-precision multiplication for public-key cryptography on embedded microprocessors," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 459–474.

[12] H. Seo and H. Kim, "Multi-precision multiplication for public-key cryptography on embedded microprocessors," in *International Workshop on Information Security Applications*. Springer, 2012, pp. 55–67.

[13] A. Karatsuba, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, 1963, pp. 595–596.

[14] H. Seo, Z. Liu, Y. Nogami, J. Choi, and H. Kim, "Hybrid Montgomery reduction," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 3, pp. 1–13, 2016.

[15] Z. Liu, K. Järvinen, W. Liu, and H. Seo, "Multiprecision multiplication on ARMv8," in *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2017, pp. 10–17.

[16] H. Seo, Z. Liu, P. Longa, and Z. Hu, "SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–20, 2018.

[17] H. Seo, P. Sanal, A. Jalali, and R. Azarderakhsh, "Optimized implementation of SIKE round 2 on 64-bit ARM Cortex-A processors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2659–2671, 2020.

[18] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Towards optimized and constant-time CSIDH on embedded devices," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2019, pp. 215–231.

[19] ARM, "ARM architecture reference manual: ARMv8, for ARMv8-A architecture profile," 2020.