International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-10 Issue-6, August 2021

Performance of Adder Architectures on Encrypted Integers

Paulin Boale Bomolo, Simon Ntumba Badibanga, Eugene Mbuyi Mukendi

***Abstract:** The fully Homomorphic encryption scheme is corner stone of privacy in an increasingly connected world. It allows to perform all kinds of computations on encrypted data. Although, time of computations is bottleneck of numerous applications of real life. In this paper, a brief description is made on the homomorphic encryption scheme TFHE of Illaria Chillota and the others. TFHE, implemented in c language in a library, improves the bootstrapping execution time of the FHEW scheme to 13 milliseconds. TFHE performs homomorphic processing on a multitude of logic gates. This variety made it possible to construct, implement five adder architectures and compare them in terms of the execution time of the bootstrapping per logic gate. In a single-processor computing environment, the Carry Look-ahead Adder completed a two-integer addition in 90 seconds, whereas the Ripple carry Adder did the same processing in 109 seconds. An improvement in processing time of 15% is observed. And, the same ratio of about 15% was obtained on four integers, respectively for 279 seconds for the first adder and 320 seconds for Wallace's dedicated adder. While in the dual-processor environment, a 50% improvement was seen on all adders in the same processing on integers. The Carry Look-ahead Adder saw his handling improved by the sum of two numbers from 90 seconds to 46 seconds and four numbers from 279 seconds to 139 seconds, respectively.*

***Keywords:** fully Homomorphic encryption, bootstrapping, logic gate, binary adder.*

I.INTRODUCTION

Homomorphic encryption performs processing on encrypted data without decrypting them. This concept remained an open problem for a long time until the breakthrough of Gentry in 2009 [4] who showed in his thesis the possibility of dealing any function on encrypted data.

In homomorphic encryption, plaintexts are encrypted by masking a value called noise and decryption consists of removing said noise to retrieve the original plaintext. Said noise increases in value after each homomorphic evaluation of an elementary operation. The somewhat homomorphic encryption scheme evaluated a limited number of various operations up to a threshold where the decryption fails. This number may be asymptotically made unlimited by the bootstrapping technique. Said technique introduced by Gentry reduces the value of noise in the resulting encrypted message.

Manuscript received on August 19, 2021.
Revised Manuscript received on August 25, 2021.
Manuscript published on August 30, 2021.

* Correspondence Author

Paulin Boale B.*, Lecturer & Ph.D, Department of Mathematics and Computers Sciences. University of Kinshasa. Congo - Kinshasa

Simon Ntumba B., Professor and head Department of Mathematic and Computers Sciences University of Kinshasa. Congo - Kinshasa

Eugene Mbuyi M, Professor, Department of Mathematic and Computers Sciences University of Kinshasa. Congo - Kinshasa

It allows a homomorphic evaluationofarbitrary circuits and including its own decryption circuit. It is very expensive in terms of time and space. Since then, several improvements have been proposed either in terms of efficiency [6][14] or by new alternative concepts [10].

Despite this, small changes have been observed until [5] which presents a very fast bootstrapping that takes place around 0.69 seconds. Said technique paves the way for applications with more complex circuits by a homomorphic universal NAND on a bit with an evaluation key of about 1 GBytes. This performance was improved by [7] and [8] by reducing the execution time to 0.1 seconds with an evaluation key of about 23 MBytes. It is implemented in a library called TFHE.

Based on this library, this paper evaluates the performance of different most well-known circuits in homomorphic additions on two or more 16-bit integers.

II.PRELIMINARY CONCEPTS

A. *Notations and symbols.*

The symbols and notations listed below will be used in the remainder of this document:

- B the set of 0,1 ;
- ai is the value of ith bit of integer a ;
- T the real torus RZ: the fractional part of a real number;
- M(N)X the set of polynomials under an abelian group M modulo XN+1: MXXN+1;
- Mn The set of vectors of (dimension) of n elements of M;
- et Mn,m the set of dimension matrices of mn elements of M.

1. *The* R-*module.*

Given R, +, × a commutative ring. A set M is a R-module if M, + is an abelian group, and if there is a Bi-distributive and homogeneous external operation. Namely, r,s ∈R et x, y ∈M, 1R. x=x, r+s.x=r.x+r.s, r.x+y=r.x+r.y, et r ×s.x=r.(s.x).

B. *The homogeneous version of the Learning problem With Errors (LWE).*

Given n ≥1 an integer, the noise ∈ R+ parameter, and a uniformly distributed secret s within a certain limit of SZn. A distribution on TnT is denoted Ds,LWEa, b. It is obtained by drawing the pair (a, b), where the left member a is chosen uniformly and randomly in Tn and the right member is an evaluation of the expression b=as+e . The error e is taken from a Gauss distribution of parameter .

- Search problem: given LWE samples, find sS;
- Decision problem: distinguish between two distributions of LWE samples and uniform and random samples from TnT.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

C. The hard problem of Learning With Errors on a Torus (TLWE).

Let be k≥1 an integer, N a power of 2, and ≥0 a noise parameter. A TLWE s ∈ BNX secret key is a vector of k polynomials ∈ R=ZXXN+1 with binary coefficients. The sample space for messages is TNX. A fresh TLWE sample ∈ TNX message with the parameter under the key s is an element a, bTNXkTNX, b ∈ TNX with a Gaussian distribution D, sα+ around +sμ . The sample is random if and only if its left member a called a mask is uniformly random in TNXk, trivial if a is fixed to 0, less noisy if =0, and homogenous if and only if =0.

- Search problem: given several TLWE samples, find their keys ∈ BNXk;
- Decision problem: distinction between a homogeneous and random TLWE sample from a uniform and random sample of TNXk.

D. The phase of a sample.

Let c= a, bTNXkTNX et sBNXk, the phase of a sample is defined by the expression sc=b-as. A phase is linear on TNXkand is kN+1-Lipschitzian for the norm if l∀ x, y ∈ TNXk+1, ‖sx-sy‖kN+1‖x-y‖

III.THE TFHE HOMOMORPHIC ENCRYPTION SCHEME

GSW is a leveled homomorphic encryption scheme that was proposed by Gentry, Sahai and Waters in [3] and has been improved in [11]. Its security is based on the error learning problem (LWE).

A. TGSW.

The Torus GSW is a generalization of the scaled invariant version of GSW. It is also extending the decomposition function to polynomials. This threshold approximation of accuracy parameter induces an improvement in execution time and memory prerequisites for additional noise.

1. Decomposition function.

Let as h∈Md,k+1TNX in (1). Dech,, vis a decomposition algorithm on h, with quality and precision if and only if for any TLWE sample v ⊂ TNXk+1, its efficient and public output gives a small vector uRdsuch that ‖u‖ et ‖uh-v‖. In addition, u.h-v must be 0 when v is uniformly distributed in TNXk+1.

1Bg 1Bgl 0 0 ⋱ 0 0 1Bg 1Bgl 1

2. TGSW sample.

Let be l et k ≥1 two integers, the noise parameter ≥0 and h the decomposition function defined in (1). Let sBNXk be a key RingLWE. C ∈ Mk+1l, k+1TNX is a fresh TGSW sample of ∈ Rh with a noise parameter if and only if C=Z+.hZ ∈ Mk+1l,k+1TNX where each row of is homogeneous TLWE sample of 0 with a gauss parameter.

Conversely, an element C ∈ Mk+1l, k+1TNX is a valid TGSW sample if and only if there exists a unique s and a unique key ∈ Rh such that each row of C-uh is a valid TLWE sample 0 for a key s. The polynomial is the message C, and denoted by msg(C).

3. Phase and error.

Let be A∈ Mk+1l, k+1TNX a TGSW sample for a secret key sBNXk by the parameter ≥0. sATNXk+1l, The noted phase A, is defined as a list of k+1l TLWE phases of each row of A. Similarly, the error of A , denoted err(A) , is defined as the list of k+1l TLWE errors in each row of A.

4. External product.

The external product . is defined as follows:

. :TGSW ×TLWE →TLWE

.→A.b= Dech,β,ϵb.A

5. Theorem 1.

Let A a valid TGSW sample of the message A and b a TLWE sample of the message B then A.b is a TLWE sample of the message A.B and ‖errA.B‖≤ k+1lNβ‖errA‖+‖A‖11+kN+‖A‖1‖errB‖ where et are the parameters used in the decomposition function Dech, ,b If .. ‖errA.B‖14 then A.B is valid TLWE sample.

6. *The internal* product.

Let be a product : TGSW ×TGSW →TGSW

7. *A, B*→A ⊠B= *b1⁝A.bk+1l=h,β,ϵ1.A⁝Dech,β,ϵk+1l.A*

With A, B two valid samples TGSW respectively of the messages A et B and bi corresponding to the ith row of B. AB is a valid TGSW sample of the message A.B and ‖errA.B‖≤ k+1lNβ‖errA‖+‖A‖11+kN+‖A‖1‖errB‖ If ‖errA.B‖14 then is a valid TGSW sample A.B.

7. Bootstrapping in the TFHE.

Theorem 1 is used to speed up bootstrapping presented in [5]. The performed optimizations reduced the size of the bootstrapping key and removed excess noise in ciphertext.

To perform bootstrapping, a sample LWE (a, b)∈Tn+1X is scaled back as a, b mod 2N using ciphertexts from its secret key sBn, the following steps must be followed:

1. Choose a phase detector testv ∈ TN a fixed polynomial whose coefficients are setting up to values that bootstrapping must return if sa, b=i2N;
2. Encode testv in a trivial TLWE sample;
3. Then, rotate the coefficients using external multiplication with TGSW ciphertexts of hidden monomials X-siai. testv rotates from a hidden phase of a, b;
4. Finally, extract the constant terms as an LWE sample.

a. Extracting LWE from TLWE.

Extracting an LWE sample from a TLWE sample consists of rewriting the polynomials in their coefficients ignoring the last N-1 coefficients of b. it provides an LWE ciphertext of constant terms of the initial or original polynomial message.

Definition 1. Let a'',b'' a sample TLWEs''with a key s''Rk , KeyExtracts'' is the vector of integers s'=coefss1''X, ……………,coefssk''X ZkN and Sampleextracta'',b'' the sample LWEa',b'∈ TkN+1 where a'=coefsa1''1X, …………,coefsak''1X and b'=b0'' the constant term of b'' Then s'a',b' (resp msga',b') is equal to s''a'',b''the constant term of resp au terme constant de =msga'',b'' and ‖Erra',b'‖‖Erra'',b''‖ and VarErra',b'VarErra'',b''.

Retrieval Number: 100.1/ijeat.F30830810621
DOI: 10.35940/ijeat.F3083.0810621
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

b. Procedure for switching keys in an LWE sample.

Given LWEs' a sample of a message T, the key switching procedure initially proposed in [9,6] outputs a sample of the same message without increasing noise. This procedure tolerates the approximation of this scheme unlike its use in other schemes.

Definition 2. Let s'0,1n', s0,1n , ∈R be a parameter and t∈N a precision parameter, the switching key KSs's,,t is a sequence of fresh samples of LWE KSi,jLWEs,si'2-j for i1,n' and j1,t.

Algorithm 2: Key switching procedure.

Input: A sample LWE a'=a1',......,an''LWEs', the switching key KSs's where s'0,1n's0,1n, t∈N and a precision parameter.

Output: an LWE sample LWEs.

1. Set up ai' a multiple close to 12l of ai', so ai'-ai'<2-t+1 ;
2. Decompose into binary each ai'=j=1tai'2-j where ai,j'0,1 ;
3. Return a,b=0,b'-i=1n'j=1tai,j'KSi,j.

c. The bootstrapping procedure.

Given an LWEs=a,b sample, said procedure constructs a ciphertext of under the same key s but with a fixed and low noise. As in [14], a TLWE sample is used as an intermediate cipher to perform a homomorphic evaluation of the phase, but here the external product of theorem 1 is used with a TGSW ciphertext of the key s.

Definition 3. Let sBn, s''BNXkand a noise parameter. The bootstrapping key BKss'', is defined as a sequence of n TGSW samples where BKiTGSWs'',si.

Algorithm 3: Bootstrapping procedure.

Input: a sample LWEa, bLWEs,, a bootstrapping key BKss'',, a switching key KSs's, where s'=Key Extracts'' and two messages 0,1T.

Output: a sample LWEs0 si sa,b ∈]-14, 14[sinon 1.

1. Set up =1+02 and '=0- ;
2. Set up b=⌊2Nb⌉ and ai=⌊2Nai⌉ for i1,n ;
3. Set up testv:=1+X+............+XN-1X-2N4.'TNX
4. Acc←Xb.o,testvTNXk+1
5. pour i de 1 à n
6. Acc←h+X-ai-1.BKi . Acc
7. Set up :=0, +Sample Extract Acc
8. Return Key Switch KS.

8. The TFHE library.

TFHE is an open source library for fully homomorphic encryption distributed under the terms of the Apache 2.0 license. It is written in C/C++ by implementing a very fast bootstrapping based on the [7,8,9].

It homomorphically evaluates 10 logical gates (AND, OR, NAND, NOR, ... etc) as well as negation NOT and The MUX gate. Each binary gate takes about 13 milliseconds which improve the [15] by a factor of 53, and the MUX gate takes about 26 CPU-milliseconds.

Bootstrapping in this library does not impose a restriction on the number of gates or even on the circuit composition compared to the [5] which does not support similar inputs.

1. Features of the TFHE library.

It is easy to use on manually made circuits and circuits automatically generated by a hardware or software utility.

From the user's point of view, this library can:

1. Generate a set of secret keys and a set of keys for the cloud. All secret keys are private, and provide encryption and decryption capability respectively. All keys for the cloud can be exported to the cloud, and allow operations to be performed on encrypted data;
2. With all the secret keys, the library is used to encrypt and decrypt the data. Encrypted data can be securely exported to the cloud to perform homomorphically secure calculations;
3. With all the cloud keys, the library can evaluate a list of binary gates homomorphically at a rate of 76 gate per second per core without decrypting them.

2. Fast Fourier Transform processors.

To run the TFHE needs at least one of the processors listed in the table below:

Table-I: FFT Processors

Name	License	Language and portability	Performance	Website
Nayuki	Mit	C and AVX	1	www.nayuki.io
spqlios	Apache 2	AVX and FMA	1	
FFTW3	Gpl	C and FORTRAN	2 - 3	www.fftw.org

In terms of performance, the FFT processor performs better than the other two. It reduces their execution times by a factor of 2 or 3.

IV. HOMOMORPHIC ADDITION OPERATIONS WITH TFHE

The plaintext space in the TFHE is Z2. The addition operation is defined in said scheme using respectively the logical gates XOR and AND. These gates are the cornerstone of the implementation of increasingly complex circuits. Addition is performing by adder. This section presents an implementation of arithmetic addition by making the full binary adder with the AND and XOR gates.

This arithmetic addition operation will be performed on integers with a size of 16 bits.

A. Adder.

The adder is a circuit that is made from two basic circuits which are the half-adder and the full adder. These are using for making four architectures of adders mentioned above.

1. Half-adder.

The half-adder is a circuit that allows the calculation of the sum s and the output carry c when adding two bits a and b.

$$s=a\oplus b \text{ et } c=ab$$

2. Full adder.

A full adder is a circuit that allows the calculation of the ith sum si and the i+1th carry ci+1 when adding two bits and an input carry of ith stage. They are ai, bi and ci includes half-adders and full adders. The difference is that a half-adder does not accept a carry while the adder accepts it.

The implementation can vary as long as the logical expressions of different implementations are equivalent. In [1], for example, the expressions of sum and carry can be written as follows:

$$c_{i+1}=a_i.b_i\oplus c_i.a_i\oplus b_i$$
$$s_i=a_i\oplus b_i\oplus c_i$$

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Where ai and bi are the ith bit of two summations, ci is the ith carry, and si is the ith sum of bits. The expression of carry may be reduced as follows:

$$c_{i+1}=a_i.b_i \oplus c_i.a_i \oplus b_i = a_i \oplus c_i.b_i \oplus c_i c_i$$

This optimized expression is found in [2]. It uses only for each bit an AND gate, and therefore a full adder of one bit at a multiplicative depth equivalent to1 (L = 1).

B. Adder Architectures.

The adder circuit has built five addition circuits which are the Ripple Carry Adder (RCA), the Carry Look-ahead Adder (CLA), Carry Save Adder (CSA) and Carry Select Adder and Wallace shift adder.

1. The carry propagation adder.

The carry propagation adders (called Ripple Carry Adder) allow to perform the addition of two binary numbers of n bits, a=an-1,an-2,…………., a0 and b=bn-1, bn-2,…………,b0, and an optional carry cin, ensuring the propagation of the carry. The result is a number of n + 1 bits, consisting of a number s=sn-1,sn-2,………..,s0 and a carry cout. The final result is obtained by waiting for the propagation of carry through the n cells of full adders. In this architecture, an adder constitutes a stage and therefore, the carry propagates from the least significant stage to the most significant stage.
The n bit carry propagation adder algorithm is constructed by n-1 full adder. This adder adds one bit at a time from less significant bits to more significant bits. The multiplicative depth is L = n – 1, for each bit except the most significant bit of the bit, a gate AND is useful and each subsequent bit depends on the preceding bit.

Algorithm 4:
Input: two n bit-encrypted integers a, b
Output: the sum s of n bits
c0 = 0
Pour i = 0 à n – 2
Faire ci+1=ai⊕ci.bi⊕cici
si=ai⊕bi⊕ci
fin faire
End For
sn-1=an-1 ⊕ bn-1 ⊕ cn-1
return s

2. The carry anticipation adder.

In a carry propagation architecture, the addition depends on the propagation of the carry through stages of the parallel adder. To reduce the propagation time and speed up the addition processing, it is possible to anticipate the output carry of each stage and to produce, from the inputs, the carry by generation or propagation. This technique is called "carry anticipation."
A carry generation occurs when a carry is generated by the full adder. A carry can only take place when the two input bits are 1 . The carry generated is denoted g and is equivalent to g =ab.
A carry propagation is created when an input carry is passed to the output carry. In a full adder, the propagation of an input carry can take place when at least one of the bits is 1. The propagated deduction denoted p and is equivalent to p = a+b. The output carry of a full adder can be expressed as a propagated carry p or as a generated carry g. The denoted csor output carry is 1 if the generated output is 1 or if the propagated output is 1 and the input carry (cen) is 1.

In other words, an output carry of 1 is generated by the full adder if a=1 et b=1 or by propagation of the adder of the input carry(a=1 ou b=1) et (cen =1). The following expression summarizes all the cases: csor =g + p.cen.
Let's illustrate this concept by applying it to a four-bit parallel adder. Stage i produces an output carry either by generating it or by propagating the internal carry to the output carry. For each stage i, it generates gi and pi propagates as follows:

- Column i produces an output carry if the inputs;ai and bi are equal to a binary 1: gi =ai .bi ;
- Column i propagates the internal carry to the output carry if one of the inputs is equal to1: pi = ai + bi ;
- The output carry of column i is given by the following expression:

$$c_i = a_i.b_i + a_i + b_i.c_{i-1} = g_i + p_i.c_{i-1}.$$

The carry anticipation adder algorithm can be described in the steps below:

- Step 1: calculate gi and pi for all columns;
- Step 2: calculate the g and p for each block of k-bits;
- Step 3: the cen input carry propagates through the k-bit block by the functions of generation and propagation of carry.

Example for a block of 4 bits (p3:0 and g3:0):
g 3:0= g3 + p3.(g2 + p2.(g1 + p1.g0)
p3:0 = p3.p2.p1.p0
In general,

$$g_{i:j} = g_i + p_i.(g_{i-1} + p_{i-1}.(g_{i-2}+p_{i-2}.g_j))$$
$$p_{i:j} = p_i.p_{i-1}.p_{i-2}.p_j$$
$$c_i = g_{i:j} + p_{i:j}.c_{j-1}$$

The complexity of the algorithm of the adder with anticipated carry respectively in time is O(n log n). The carry anticipation adder of n is faster than the carry propagation adder which has respectively a time and space complexity of O(n).

3. Carry Save Adder.

Carry Save Adder (CSA) perform the addition function by dealing with the intermediate carry as an output, and without propagating it through the next cell. The carry of each stage is thus "saved". The result is composed of two numbers of n bits: S for the sum and C for the carry. The architecture of this adder is a linear arrangement of full adders. An additional calculation must be made to obtain the result.
The carry save adder is a set of k full adders paralleled without any horizontal connection. The main feature of this circuit is the addition of three numbers a, b and c to produce two numbers c' and s such that c' + s = a+b+c.
Given a=40, b=25 and c=20, c' and s are calculated as follows:

Table-II: carry saver adder of three numbers of 8 bits.

a	=	40	=		1	0	1	0	0	0
b	=	25	=		0	1	1	0	0	1
c	=	20	=		0	1	0	1	0	0
S	=	37	=		1	0	0	1	0	1
c'	=	48	=	0	1	1	0	0	0	

The ith bit of the sum Si and (i + 1)th bit of the report ci+1 is calculated using the expressions given below:
si=ai⊕bi⊕ci
ci+1=ai.bi⊕ci.ai⊕bi=
ai⊕ci.bi⊕cici

Retrieval Number: 100.1/ijeat.F30830810621
DOI: 10.35940/ijeat.F3083.0810621
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

In other words, a Carry Save Adder circuit is a full adder cell with three data inputs instead of two inputs plus the previous carry.
And to determine the noted r result of adding three numbers, the following steps are performed:

- For the stage of the adder with the least significant value: r0= s0;
- For the adder stage whose value directly precedes the least significant, the expressions are used and r1 = s1+c'1c2'=s1c'1;
- For the other stages, the expressions of the fully adder are used;
- For the stage of the adder with the most significant value: rn-1 = cn-1 +c'n-1.

5. The selective carry adder.

A Carry Select Adder is a logical and arithmetic combinatorial circuit that sums two numbers of n-bits and outputs their sums of n-bits and a carry bit.
It offers a different design than that of a carry propagation adder. It does not propagate the carry through the full adders. Thus, the addition time is reduced.
It is a circuit composed of two adders with parallel propagation of n-bits and a multiplexer for the selection of the outgoing sum. To perform an addition between two numbers of n-bits, two propagation adders receive at all stages respectively an incoming carry at 1 and an incoming carry at 0, once the effective carry is generated a simple active selection of the appropriate outgoing sum.

6. Wallace tree adder.

Wallace tree adders are composed of a tree structure of carry save adders, and a Ripple Carry Adder. This configuration is a very fast multi-operand architecture. The following expressions give the example of adding 4 operands denoted a, b, c, and d:
First stage:

set10= a0⊕b0⊕c0
cet11=a0c0.b0c0c0

Second stage:

set20= set10⊕cet10⊕d0
cet21=set10d0.cet10d0d0

Third stage:

s0= set20⊕cet20⊕cet30
cet31=set20cet30.cet20cet30cet30

V. EXPERIMENTAL RESULT

This section reports the experimental results of our implementation described above.

A. Setting up parameters.

To perform this experiment, the provided default security settings were used without any changes. The library provides an API that implements the majority of logical gates. the gates below have built the architecture of different adders listed above:

1. Homomorphic assignment function: *void boots CONSTANT (Lwe Sample* result, int value, const TFhe Gate Bootstrapping Cloud Key Set* bk);*
2. Function of copying one variable into another: *void boots COPY (Lwe Sample* result, const Lwe Sample* ca, const TFhe Gate Bootstrapping Cloud Key Set* bk);*
3. Logical function to reverse a boolean value: *void boots NOT (Lwe Sample* result, const Lwe Sample* ca, const TFhe Gate Bootstrapping Cloud Key Set* bk);*
4. Two-bit multiplication logic function: *void boots AND (Lwe Sample* result, const Lwe Sample* ca, const Lwe Sample* cb, const TFhe Gate Bootstrapping Cloud Key Set* bk);*
5. Two-bit addition logic function: *void boots XOR (Lwe Sample* result, const Lwe Sample* ca, const Lwe Sample* cb, const TFhe Gate Bootstrapping Cloud Key Set* bk);*

B. Performance and interpretation.

The implementations were tested on three environments that has a **RAM of 4 Gigabytes.** In Table 1, the column represents the type of processor used during the experiment and the row when it the adder type. The intersection between the row and the column represents the time it takes to perform an addition operation on two numbers of 16-bit numbers, respectively.

Table-III: Performance of adders in adding two numbers.

Duration(s)	AMD E1-2100 APU with Radeon ™ HD Graphics 1000 Mhz	Intel® Core™ i5-3210 CPU @ 2.50 Ghz	Intel ® Xeon ® CPU 5120 @ 1.86 Ghz(2)
RCA	131(2)	109(2)	55(2)
CLA	109(2)	90(2)	46(2)
CSA	**247(3)**	**205(3)**	**105(3)**
CSSA	393(2)	325(2)	167(2)

In Table 1, the more CPU capacity increases, the shorter the execution time. The execution time of the addition of two numbers on a carry propagation adder architecture is reduced by 16% from processor 1 to processor 2, from processor 2 to processor 3 and by 58% from processor 1 to processor 3, respectively.
The best architecture in terms of performing the addition of two numbers is the anticipated carry. It improves the execution time of the addition on two numbers respectively by 16% on average on all processors of the carry propagation technique and by 72% of the selective carry technique.

Table-IV: Performance of adders on the sum of four numbers.

Duration(s)	AMD E1-2100 APU with Radeon ™ HD Graphics 1000 Mhz	Intel® Core™ i5-3210 CPU @ 2.50 Ghz	Intel ® Xeon ® CPU 5120 @ 1.86 Ghz(2)
RCA	393(4)	333(4)	167(4)

Retrieval Number: 100.1/ijeat.F30830810621
DOI: 10.35940/ijeat.F3083.0810621
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

CLA	326(4)	279(4)	139(4)
CSA	**378(4)**	**322(4)**	**161(4)**
WALLACE	**377(4)**	**320(4)**	**161(4)**

The pipeline anticipated carry architecture for the addition of four numbers is better than the CSA and WALLACE dedicated architectures. Indeed, a reduction of 15% is noted between it and dedicated architectures regardless of the type of processor. The multiprocessor environment gives encouraging results compared to the single processor environment with a 50% reduction in execution time.

VI. CONCLUSION

This paper applies the bootstrapping is implemented by Illaria Chillota and al. which performs a two-bit homomorphic logic operation in 13 milliseconds and is generalized to multiple input and multiple output adder architectures. This extension took advantage of the efficiency of these schemes to handle additions over two integers and four integers of 16-bit. The implementation of these adder architectures in the c language has given promising results on three computing environments. Indeed, regardless of the environment used single or multiprocessor, the Carry Look-ahead Adder architecture applies a reduction coefficient to the execution time compared to other architectures. In addition, the multiprocessor environment has been a useful in the extent of addition on encrypted data because it reduces by half the execution time of an addition on several encrypted numbers. Although the execution time of this operation is still in the order of minutes (about 3 minutes). It would be useful to explore the parallelism per Central Processing Unit or per Graphics Processing Unit for more performance.

REFERENCES

1. [YG15] Chen, Y., Gong, G.: Integer arithmetic over ciphertext and homomorphic data aggregation. In: Proceedings of 2015 IEEE Conference on Communications and Network Security, pp. 628–632. IEEE, Piscataway, NJ (2015);
2. [KSS09] Kolesnikov, V., Sadeghi, A.R. Scheinder, T.: Improved garbled circuit building blocks and application to auctions and computing minima. In: Garay, J.A., Miyaji, A, Otsuka, A. (eds) CANS 2009, CNSL, vol. 5888, pp. 1–20. Springer Berlin (2009);
3. [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". In: CRYPTO 2013, Part I. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. CNSL. Springer, Heidelberg, Aug. 2013, pp. 75–92. doi: 10.1007/978-3-642-40041-4_5;
4. [GEN09] Craig Gentry "A fully homomorphic encryption scheme". crypto.stanford.edu/craig. PhD thesis. Stanford University, 2009;
5. [DM15] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: EUROCRYPT 2015, Part I. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. CNSL. Springer, Heidelberg, Apr. 2015, pp. 617–640. DOI: 10.1007/978-3-662-46800-5_24;
6. [DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "Fully Homomorphic Encryption over the Integers". In: EUROCRYPT 2010. Ed. By Henri Gilbert. Vol. 6110. CNSL. Springer, Heidelberg, May 2010, pp. 24–43.
7. [CGGI16a] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption Library over the Torus. https://github. com /tfhe/tfhe. 2016;
8. [CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds". In: ASIACRYPT 2016, Part I. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. CNSL. Springer, Heidelberg, Dec. 2016, pp. 3–33. doi: 10.1007/978-3-662-53887-6_1;
9. [CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE". In: ASIACRYPT 2017, Part I. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. CNSL. Springer, Heidelberg, Dec. 2017, pp. 377– 408;
10. [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: ITCS 2012. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325;
11. [AP14] Jacob Alperin-Sheriff and Chris Peikert. "Faster Bootstrapping with Polynomial Error". In: CRYPTO 2014, Part I. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. CNSL. Springer, Heidelberg, Aug. 2014, pp. 297–314. doi: 10.1007/978-3-662-44371-2_17;
12. [AAR78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. "On Data Banks and Privacy Homomorphisms". In: Foundations of Secure Computation, Academia Press (1978), pp. 169–179;
13. [Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: 37th ACM STOC. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93;
14. [ZIMMERMANN] Binary Adder Architectures for Cell-Based VLSI and their Synthesis. PhD Thesis Swiss Federal lnstitute of Technology Zurich 1998.

AUTHOR PROFILES

Paulin Boale B. is senior lecturer and PhD Student at university of Kinshasa in Mathematics and Computers sciences department. his field of research is cryptography, in particular homomorphic cryptography. he works to improve algorithms in everyday applications. he contributed to the publication of articles respectively in the journal IJCSI and IJSR such as « Study of Master-Slave Database replication in distributed database », IJCSI , 2011.

Simon Ntumba B. is professor and head of Mathematic and computers sciences department of the University of Kinshasa. As publications, Author of many publications, such as: "Enhanced Parallel Skyline on multi-core architecture with lax Memory space Cost", IJCSI, volume 13, Issue 5, September 2016, Data mart approach for stock management model with a calendar under budgetary constraint, IJCSI, volume 15, Issue 5, September 2018, Poster et the 2nd International conference on Big Data Analysis and Data Mining, San Antonio, USA, 30 november- 01 December 2015 "; Data Mart Approach for Stock Management Model with a calendar Under Budgetary constraint, IJCSI, volume 15, Issue 5, September 2016,

Eugene Mbuyi M, is professor at the Mathematic and Computers Sciences department of the University of Kinshasa. Director of informatics laboratory of the faculty of sciences at the university of Kinshasa. He is author of many articles in many scientific journals like in IJCSI . Poster et the 2nd International conference on Big Data Analysis and Data Mining, San Antonio, USA, 30 november- 01 December 2015.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)