Accelerating Homomorphic Encryption and Its Applications on GPU

Erkay Savaş

Sabancı University

Sabancı Üniversitesi

enCRYPTON

Outline

- GPU Basics
- Homomorphic Encryption Schemes
- Ring Multiplication with NTT
- GPU Implementation
- Timing Results
- Concluding Remarks

GPU Basics – Why GPU?



- Number of parallel running threads
  - CPU: tens
  - GPU: thousands and tens of thousands
- Fast memory & High bandwidth communication with host device
- Easy integration & code development
- General purpose computing via GP-GPU (CUDA)

GPU Architecture

- Two parts
  - GPU chip
    - Streaming multiprocessors
    - **On-chip memory**: L1 cache (for **shared memory**), L2 cache, registers
  - off-chip memory (GDDR)
    - **Global memory**

GPU Chip & SM

- GPU
  - Streaming multiprocessors (SM) capable of running 1000+ threads
  - L1 cache for shared memory & registers

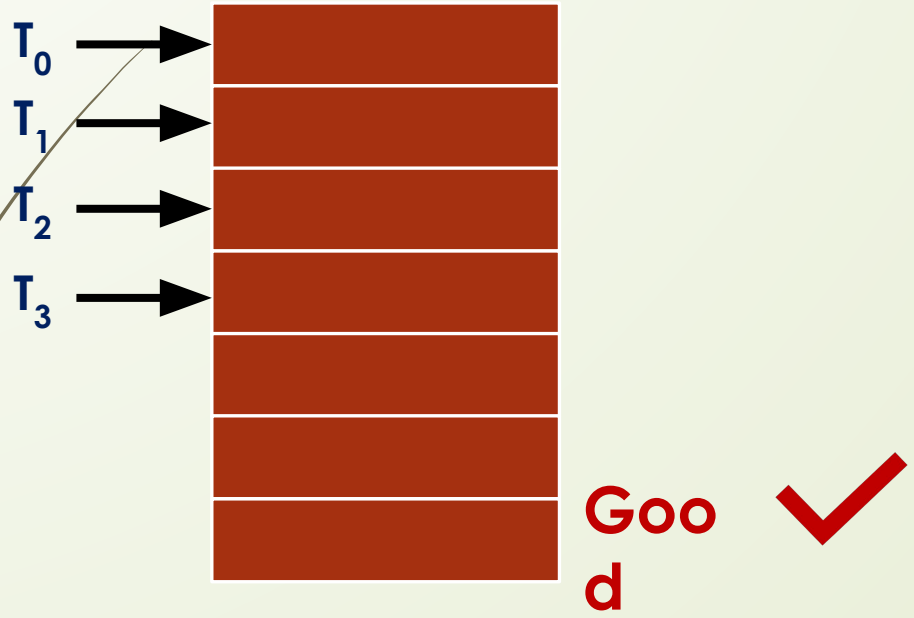
Feature	Capacity
Maximum number of **blocks** per SM	16
Maximum number of **warps** per SM	48
Maximum number of **threads** per SM	1536
Number of 32-bit registers per SM	64 K
Maximum number of 32-bit registers per thread	255
Maximum amount of **shared memory** per SM	100 KB

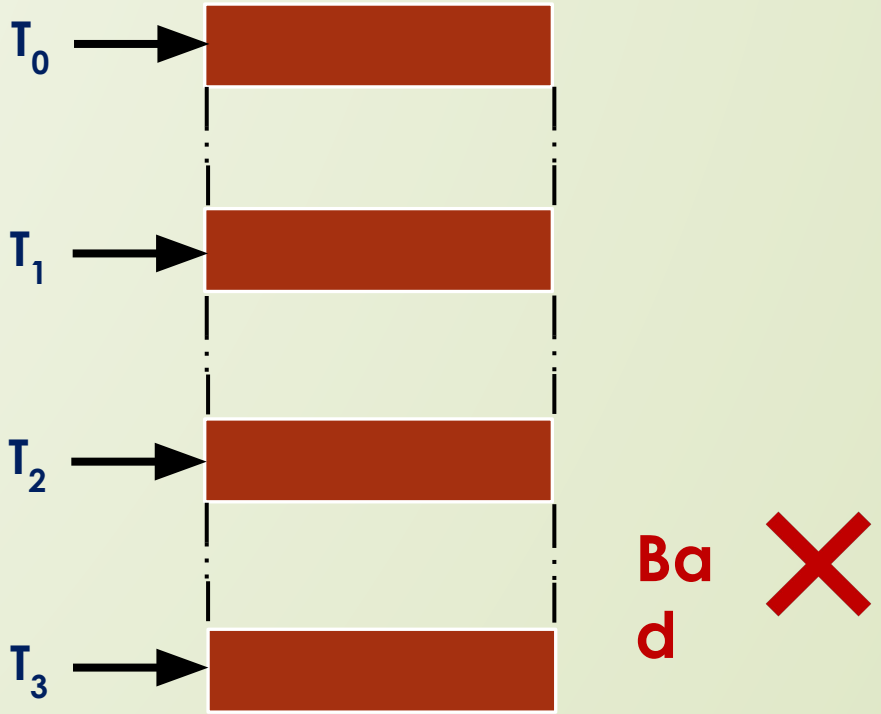
How GPU computation Organized

- **Kernel:** a function executed on GPU device
- **Block:** a group of threads **accessing the same shared memory**
  - maximum number of threads in a block is 1024
  - can synchronize via fast shared memory
  - can be indexed one-, two- or three-dimensional (**block shape**)
- **Grid:** a group of thread blocks
  - computation may require multiple blocks
  - only synchronize via slow global memory (requires terminating the kernel)
  - can be indexed one-, two- or three-dimensional (**grid shape**)

Coalesced Access

- Cache line & memory block: 128 B
  - Threads in a block access the consecutive addresses in the global memory





Occupancy

- Good utilization of GPU resources (threads, shared memory, registers)
  - maximum number of threads in a block: 1024
  - Smaller block sizes may increase occupancy

Feature	Capacity
Maximum number of blocks per SM	16
Maximum number of warps per SM	48
Maximum number of threads per SM	**1536**
Number of 32-bit registers per SM	64 K
Maximum number of 32-bit registers per thread	255
Maximum amount of shared memory per SM	100 KB

Various GPU DEVICES

FEATURES	RTX 3060Ti	RTX A6000	RTX 3090Ti	Tesla V100	A100	RTX 4090
Architecture	Ampere	Ampere	Ampere	Volta	Ampere	Ampere
Mem. Size	8 GB	48 GB	24 GB	16 GB	40 GB	24 GB
Mem. Type	GDDR6	GDDR6	GDDR6X	HBM2	HBM2e	GDDR6X
Mem. Bus	256 Bit	384 Bit	384 Bit	4096 Bit	5120 Bit	384 Bit
Bandwidth	448.0 GB/s	768.0 GB/s	1008.0 GB/s	897.0 GB/s	1555 GB/s	1008 GB/s
Boost Clock	1665 MHz	1800 MHz	1860 MHz	1380 MHz	1410 MHz	2520 MHz
Power*	550 W	700 W	850 W	700 W	600 W	850 W
Thread Count	4864	10752	10752	5120	6912	16384
L1 Cache	128 KB	128 KB	128 KB	128 KB	192 KB	128 KB
L2 Cache	4 MB	6 MB	6 MB	6 MB	40 MB	72 MB

*** suggested power supply unit capacity**

Homomorphic Encryption Schemes

- Fully or Somewhat Homomorphic Encryption Schemes (FHE or SWHE)
- Various schemes are proposed such as **BFV**, **CKKS**, **BGV**, **T-FHE**, etc.
- Their security based on hardness of **the lattice problems** (LWE, RLWE)
- RLWE hardness assumptions

RLWE hardness assumptions

- **Search problem:**
  - $R = Z[x]/\langle \Phi_m(x) \rangle$, where $\Phi_m(x) = x^n + 1$ if m is a power of 2 and $n = m/2$
  - $R_q = R/qR$
  - $a \leftarrow Rq$, $e \leftarrow \chi_{err}$, and $s \leftarrow \chi_{key}$
  - $b = as + e$
    - Given a and b, it is computationally infeasible to find s
    - (a, b): public key, s: secret key

Multiplication in R_q

- Polynomial multiplication, followed by the reduction with $\Phi_m(x)$
- But the degrees is quite high ($n > 2048$), q can get very large ($\log_2 q > 100$)
- Classical polynomial multiplication is prohibitively slow $\boldsymbol{(O(n^2))}$
  - c(x) = (a(x) × b(x) mod q) mod $\Phi_m(x)$
- Solution: NTT-based multiplication $\boldsymbol{(O(n\log n))}$
  - c(x) = iNTT(NTT(a) $\otimes$ NTT(b) mod q)

component-wise multiplication

NTT: number theoretic transform

- $\Phi_m(x) = x^n + 1 = \prod_{i=0}^{n-1}\left(x - \psi^{2i+1}\right) \bmod q$,
  where ψ is primitive 2n-th root of unity $(\psi^{2n} \bmod q = 1, \psi^n \bmod q = -1)$
  - $R_q = Z_q[x]/\langle \Phi_m(x) \rangle \approx Z_q[x]/\langle x - \psi \rangle \times Z_q[x]/\langle x - \psi^3 \rangle \times \cdots \times Z_q[x]/\langle x - \psi^{2n-1} \rangle$
- $a \times b$
- If a and b are reduced by each $\left(x - \psi^{2i+1}\right)$ then multiplication in R_q becomes component-wise multiplication of a and b reduced components

Computing NTT

- $(\psi^{2n} \bmod q = 1, \psi^{n} \bmod q = -1)$

- $x^n + 1 = x^n - \psi^n = (x^{n/2} - \psi^{n/2})(x^{n/2} + \psi^{n/2}) \bmod q$
- $a = a_0 + a_1 x + \cdots + a_{n/2-1} x^{n/2-1} + a_{n/2} x^{n/2} + a_{n/2+1} x^{n/2+1} \cdots + a_{n-1} x^{n-1}$
- Reduced by $(x^{n/2} - \psi^{n/2})$ ➔ $a_L = (a_0 + a_{n/2}\,\psi^{n/2}) + \cdots + (a_{n/2-1} + a_{n-1}\,\psi^{n/2}) x^{n/2-1}$
- Reduced by $(x^{n/2} + \psi^{n/2})$ ➔ $a_H = (a_0 - a_{n/2}\,\psi^{n/2}) + \cdots + (a_{n/2-1} - a_{n-1}\,\psi^{n/2}) x^{n/2-1}$
- **Cooley-Tukey Butterfly Circuit: $a_i + a_{i+n/2}\,\psi^{n/2}$ and $a_i - a_{i+n/2}\,\psi^{n/2}$**
- The recursion continues as n is a power of 2

$$x^{n/2} + \psi^{n/2} = x^{n/2} - \psi^n \psi^{n/2} = (x^{n/4} - \psi^{n/2}\psi^{n/4})\ (x^{n/4} + \psi^{n/2}\psi^{n/4})$$

Computing inverse NTT

- Reduced by $(x^{n/2} - \psi^{n/2})$ ➔ $a_L = (a_0 + a_{n/2}\,\psi^{n/2}) + \cdots + (a_{n/2-1} + a_{n-1}\,\psi^{n/2})x^{n/2-1}$
- Reduced by $(x^{n/2} + \psi^{n/2})$ ➔ $a_H = (a_0 - a_{n/2}\,\psi^{n/2}) + \cdots + (a_{n/2-1} - a_{n-1}\,\psi^{n/2})x^{n/2-1}$
- Combine them to get the original polynomial
- $(a_L + a_H)2^{-1} \bmod q$ ➔ $a_0, a_1, \cdots, a_{n/2-1}$
- $(a_L - a_H)\psi^{-n/2}2^{-1} \bmod q$ ➔ $a_{n/2}, a_{n/2+1}, \cdots, a_{n-1}$
- Gentleman-Sande Butterfly Circuit: $a_i + a_{i+n/2}$ and $(a_i - a_{i+n/2})\psi^{n/2}$

enCRYPTON

Algorithms

Sabancı Üniversitesi

Merge In-place Forward NTT

Input: $a \in R_q, n, q$ and

Input: $\psi_{br}[i] = \psi^{br(i)}$: powers of ψ stored in bit reverse order

Output: $\bar{a} = \text{NTT}(a)$: In bit reverse order

1: $t \leftarrow n; m \leftarrow 1;$
2: **do**
3: $t \leftarrow t/2$
4: **for** i **from** 0 **to** m **do**
5: $j_1 \leftarrow 2it;\ j_2 \leftarrow j_1 + t - 1;$
6: **for** j **from** j_1 **to** j_2+1 **do**
7: $\boldsymbol{U \leftarrow a_j; V \leftarrow a_{j+t}\psi_{br}[m+i] \bmod q;}$
8: $\boldsymbol{a_j \leftarrow U + V \bmod q;\ a_{j+t} \leftarrow U - V \bmod q;}$
9: **end for**
10: **end for**
11: $m \leftarrow 2m$
12: **while** $m < n$
13: $\bar{a} \leftarrow a$

Merge In-place Inverse NTT

Input: $\bar{a} = \text{NTT}(a)$: In bit reverse order; n, q

Input: $\psi_{rev}[i] = \psi^{-br(i)} \bmod q$

Output: $\bar{a} \in R_q$

1: $t \leftarrow 1; m \leftarrow n;$
2: **do**
3: $j_1 \leftarrow 0; h \leftarrow m/2;$
4: **for** i **from** 0 **to** h **do**
5: $j_2 \leftarrow j_1 + t - 1$
6: **for** j **from** j_1 **to** $j_2 + 1$ **do**
7: $U \leftarrow \bar{a}_j; V \leftarrow \bar{a}_{j+t};$
8: $\boldsymbol{\bar{a}_j \leftarrow U + V \bmod q}$
9: $\boldsymbol{\bar{a}_{j+t} \leftarrow (U - V)\psi_{rev}[h+i] \bmod q;}$
10: **end for**
11: $j_1 \leftarrow j_1 + 2t$
12: **end for**
13: $t \leftarrow 2t; m \leftarrow m/2$
14: **while** $t < n$
15: **for** i **from** 0 **to** $h - 1$ **do**
16: $a_i \leftarrow n^{-1}\bar{a}_i \bmod q$
17: **end for**

Example: $n = 32$

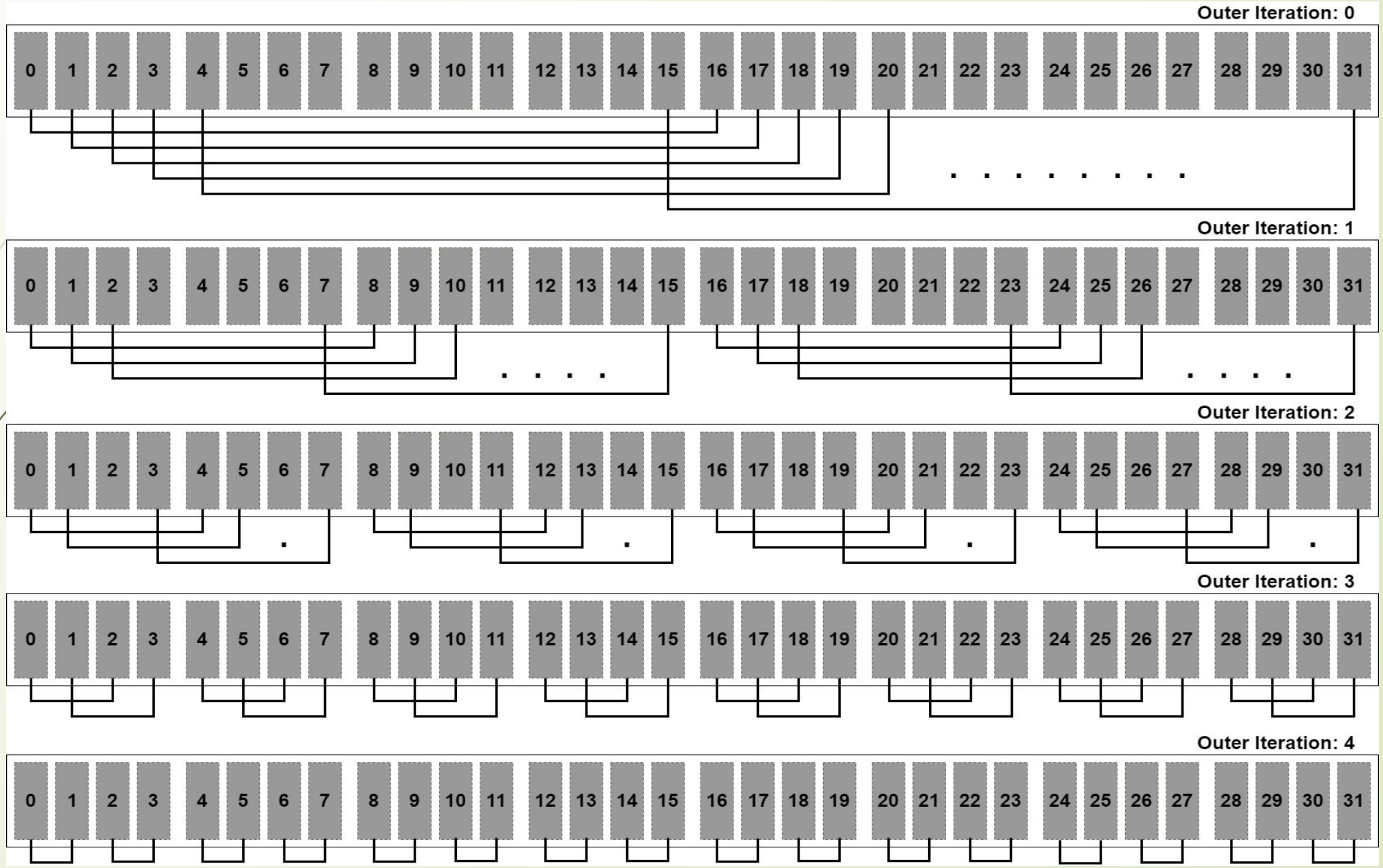
one NTT

two NTTs

four NTTs

8 NTTs

16 NTTs



GPU Algorithms for NTT

- **Mission:** perform NTT for large ring dimensions ($n=2^{12}$ to 2^{24})
  - many blocks should run (bDim ≤ 1024)
  - When n ≤ 2048, one block suffices
- **Goal:** strike a balance between potentially conflicting objectives
  - Increase occupancy
  - Better use of coalesced access to global memory
  - minimizing number of kernels used for thread synchronization
  - good distribution of outer loop iterations among the kernels
  - …

Two GPU Algorithms: NTT host

NTT HOST

Input: $A[n], PsiTable[n], n, q, mbd = 1024$

Output: $A[n]$: In bit reverse order

1: $\{bDim, kc, koc\} \leftarrow Partition(n, mbd = 1024)$

2: $bc \leftarrow n/(2bDim)$; $olc \leftarrow log_2(n)$; $oc \leftarrow -1$

3: **for** i **from** 0 **to** $kc - 1$ **do**

4: $\quad oc \leftarrow oc + koc[i]; olc \leftarrow olc - koc[i]; ko[i] \leftarrow 2^{oc}$

5: $\quad$ **if** i = 0 **then**

6: $\quad\quad kgs[i] \leftarrow [bc, 1]; kbs[i] \leftarrow [bDim/ko[i], ko[i]];$

7: $\quad$ **else**

8: $\quad\quad kgs[i] \leftarrow [2^{olc}, bc/2^{olc}]$

9: $\quad\quad kbs[i][1] \leftarrow (2ko[i-1])/kgs[i][1]$

10: $\quad\quad kbs[i][0] \leftarrow bDim/kbs[i][1]$

NTT HOST

...

11: $m \leftarrow 1$

12: **for** i **from** $kc - 1$ **downto** 0 **do**

13: $\quad$ **dim 3 B**$(kgs[i][0], kgs[i][1])$

14: $\quad$ ***dim* 3 T**$(kbs[i][0], kbs[i][1])$

15: $\quad$ **NTT** $\ll B, T \gg (A, PsiTable, m, ko[i], koc[i], q)$

16: $\quad m \leftarrow m \times 2^{koc[i]}$

kc: number of kernels

koc: number of iterations in kernels

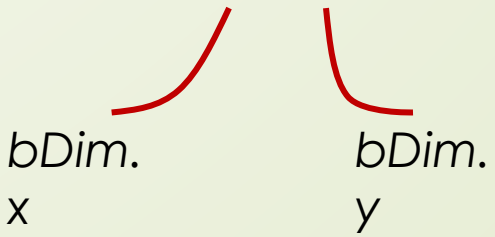
bDim: block dimension

kgs: kernel grid shape

kbs: kernel block shape

example

- $n=2^{24}$, *bDim*=1024
- *bc* ←8192
- *kc* ← 3
- *koc* ← [11, 11, 2]
  - last kernel will always have 2*bDim* outer loop iteration
- *kgs* ← [[8192, 1], [4, 2048], [1, 8192]]
- *kbs* ← [[1, 1024], **[1024, 1]**, [2, 512]]



example

- Idea is that threads in a block will share vector elements in all iterations of the kernel
- Previous example:
  - $n=2^{24}$, *bDim*=1024
  - 1st kernel: [2, 512]:
    - two groups of threads of 512 each; threads in each group accessing consecutive addresses in the global memory
  - 2nd kernel: [1024, 1]: very fragmented access to the global memory

example: affect of no of iterations in a kernel

- Previous example:
  - $n=2^{24}$, *bDim*=1024
    - *koc* ← [11, 11, 2]
  - **koc ← [11, 7, 6]**
  - *kgs* ← [[8192, 1], [64, 128], [1, 8192]]
  - *kbs* ← [[1, 1024], [64, 16], [32, 32]]

example: affect of block size

- $n=2^{17}$

bDim	koc	RTX 3060 Ti	RTX 4090
1024	[11,6]	28.8 µs	13.1 µs
256	[9,8]	25.0 µs	11.2 µs

Two GPU Algorithms: NTT Kernel

NTT KERNEL

Input: $A[n], PsiTable[n], m, ko, koc, q$

Output: $A[n]$: In bit reverse order

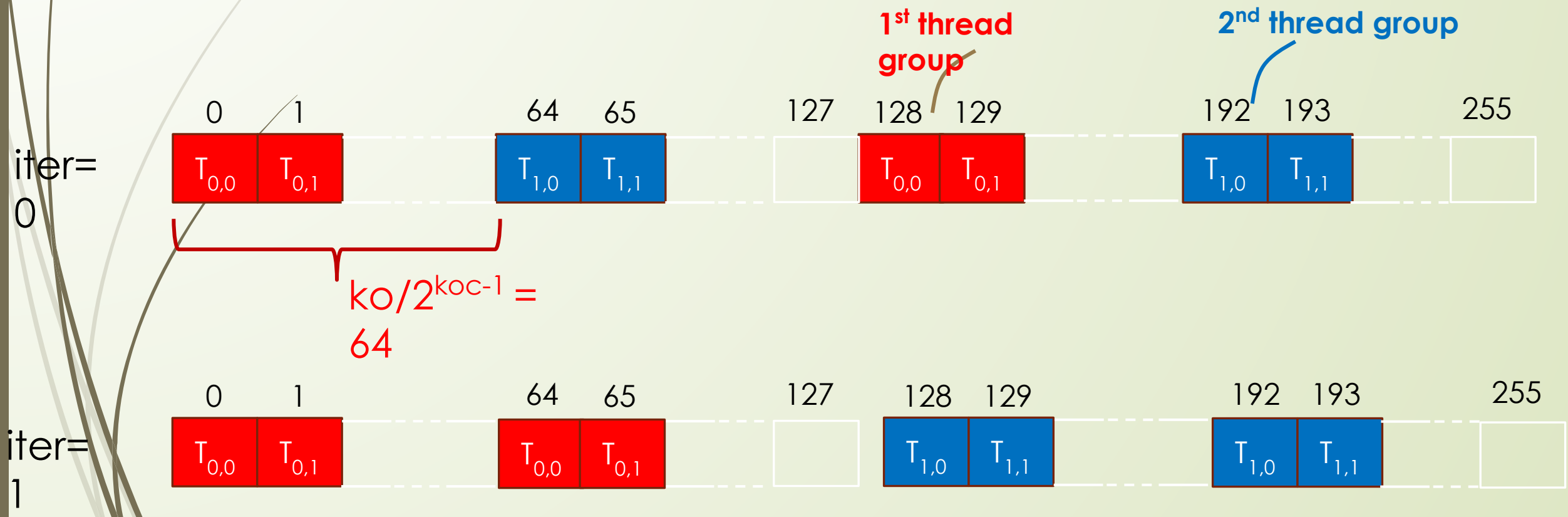
1: $t_1 \leftarrow bDim.x \times bDim.y$
2: $l_1 \leftarrow tID.x \times ko/2^{koc-1}$
3: $l_2 \leftarrow bDim.y \times bID.y$
4: $l_3 \leftarrow 2 \times ko \times bID.x$
5: $GAddr \leftarrow tID.y + l_1 + l_2 + l_3$
6: $SAddr \leftarrow tID.y + tID.x \times bDim.y$
7: $PsiAddr \leftarrow tID.y + l_1 + l_2 + l_3/2$
8: $\text{offset}_G \leftarrow ko$; $\text{offset}_S \leftarrow bDim.x \times bDim.y$
9: $\mathbf{SMEM}[SAddr] \leftarrow A[GAddr]$
10: $\mathbf{SMEM}[SAddr + \text{offset}_S] \leftarrow A[GAddr + \text{offset}_G]$

NTT KERNEL

...
11: **for** i **from** 0 **to** $koc - 1$ **do**
12: $u \leftarrow \lfloor SAddr/t_1 \rfloor \times t_1 + SAddr$
13: $PsiIn \leftarrow m + \lfloor PsiAddr/ko \rfloor$
14: $\mathbf{CT}(\mathbf{SMEM}[u], \mathbf{SMEM}[u + t_1], PsiTable[PsiIn], \mathbf{q})$
15: **syncthreads**()
16: $m \leftarrow m \times 2; t_1 \leftarrow t_1/2; ko \leftarrow ko/2$
17: $A[GAddr] \leftarrow \mathbf{SMEM}[SAddr]$
18: $A[GAddr + \text{offset}_G] \leftarrow \mathbf{SMEM}[SAddr + \text{offset}_S]$

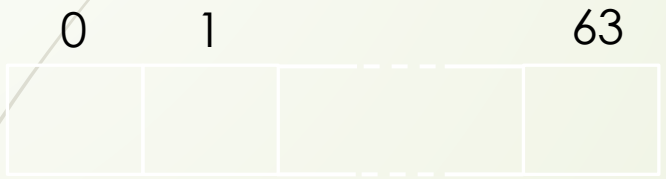
Example

- $n = 256, bDim = 4, bc = 32, koc = [3,3,\mathbf{2}], ko = [4,32,\mathbf{128}]$
- $kgs = [[32,1],[4,8],[\mathbf{1},\mathbf{32}]], kbs = [[1,4],[4,1],[\mathbf{2},\mathbf{2}]]$
- kernel = 0

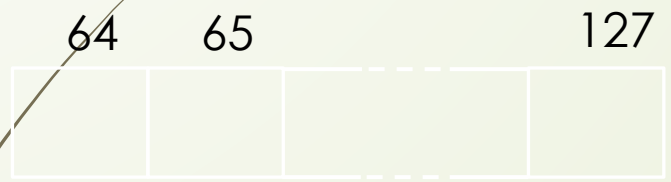


example: kernel = 1

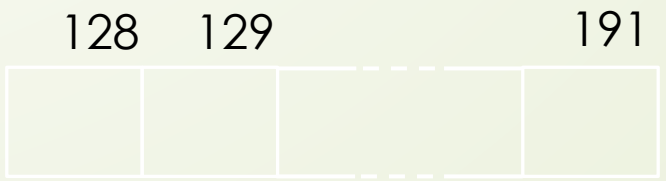
- $n = 256, bDim = 4, bc = 32, koc = [3, 3, 2], ko = [4, 32, 128]$
- $kgs = [[32, 1], [4, 8], [1,32]], kbs = [[1,4], [4, 1], [2,2]]$



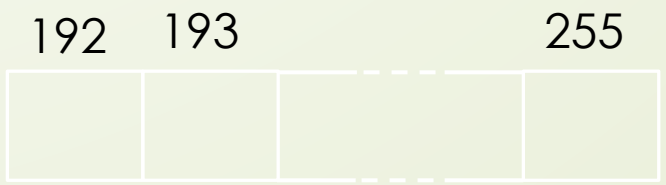
$kgs = [0, 0 \dots 7]$



$kgs = [1, 0 \dots 7]$



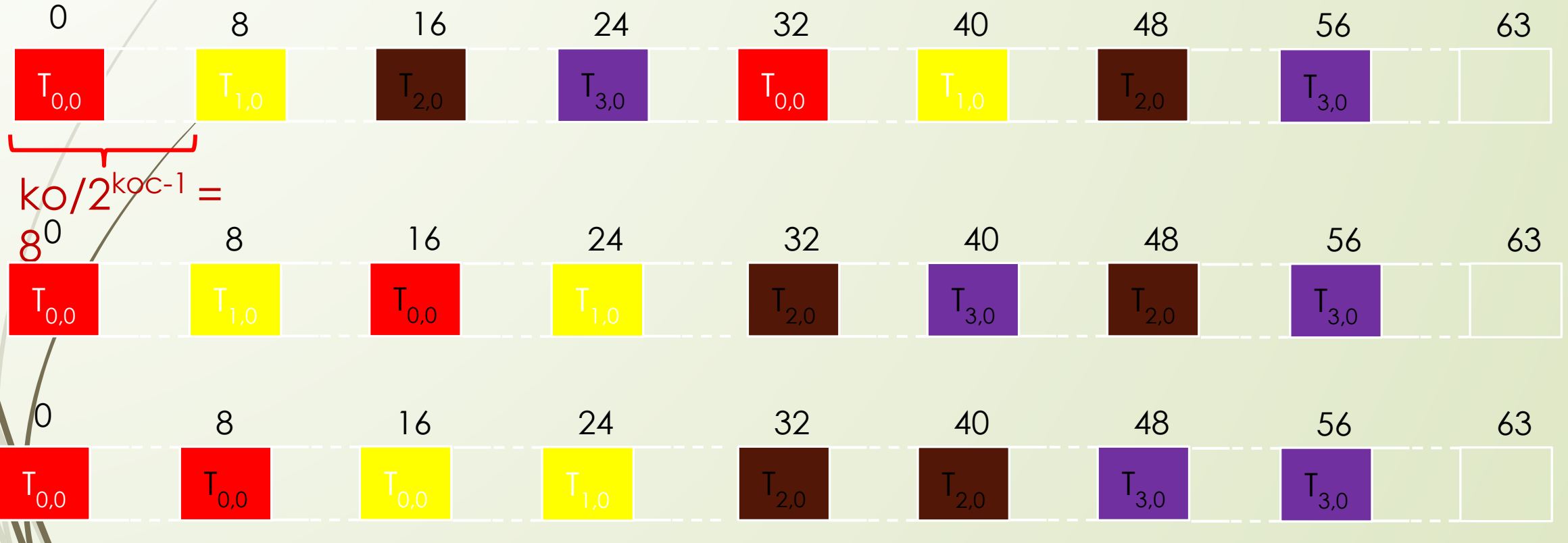
$kgs = [2, 0 \dots 7]$



$kgs = [3, 0 \dots 7]$

example: kernel = 1

- $n = 256, bDim = 4, bc = 32, koc = [3,3,2], ko = [4,32,128]$
- $kgs = [[32,1],[4,8],[1,32]], kbs = [[1,4],[4,1],[2,2]]$
- **1st grid (*bID*=[0,0])**



example: FIRST 3 BLOCKS

bID	tID	GAddr	SAddr	Corres. GAddr	Iteration
[0,0]	[0,0]	[0,128]	[0,4]	[0,128]	0
			[0,2]	[0,64]	1
	[0,1]	[1,129]	[1,5]	[1,129]	0
			[3,5]	[1,65]	1
	…				
[0,1]	[0,0]	[2,130]	[0,4]	[2,130]	0
			[0,2]	[2,66]	1
	[0,1]	[3,131]	[1,5]	[1,129]	0
			[3,5]	[3,67]	1
	…				
[0,2]	[0,0]	[2,130]	[0,4]	[4,132]	0
			[0,2]	[4,68]	1
	[0,1]	[3,131]	[1,5]	[5,133]	0
			[3,5]	[5,69]	1

$n = 256, bDim = 4,$
$bc = 32, koc = 2,$
$ko = 32$
$kgs = [1,32],$
$kbs = [2,2]$

effect of kernel count

n	bDim	koc	bDim.y	RTX 3060 Ti	RTX 4090
2^{18}	256	[9,9]	256, 1	58.3 µs	20.6 µs
	256	[9,5,4]	256, 32, 16	38.4 µs	14.0 µs

Implementation Results: GPU devices

FEATURES	RTX 3060Ti	A100	RTX 4090
Cumpute capability	8.6	8.6	8.6
Architecture	Ampere	Ampere	Ampere
Mem. Size	8 GB	40 GB	24 GB
Mem. Type	GDDR6	HBM2e	GDDR6X
Mem. Bus	256 Bit	5120 Bit	384 Bit
Bandwidth	448.0 GB/s	1555 GB/s	1008 GB/s
Boost Clock	1665 MHz	1410 MHz	2520 MHz
Power*	550 W	600 W	850 W
Thread Count	4864	6912	16384
L1 Cache	128 KB	192 KB	128 KB
L2 Cache	4 MB	40 MB	72 MB

enCRYPTON

Sabancı Üniversitesi

Implementation Results: single NTT

GPU	$\log_2 n$	$\log_2 q$	t.w. (merge)	t.w. (4-step)	[3]	[8]
RTX 3060Ti	12	≈64	7.24 µs	7.57 µs	14.0 µs	-
	13	≈64	7.5 µs	7.74 µs	14.9 µs	-
	14	≈64	9.51 µs	8.45 µs	19.1 µs	-
	15	≈64	10.93 µs	9.94 µs	35.9 µs	-
	16	≈64	13.8 µs	13.65 µs	-	-
A100	12	≈64	11.9 µs	12.3 µs	-	-
	13	≈64	11.9 µs	12.8 µs	-	-
	14	≈64	13.1 µs	13.4 µs	-	13.3 µs
	15	≈64	13.8 µs	14.0 µs	-	-
	16	≈64	21.6 µs	15.7 µs	-	16.5 µs
RTX 4090	12	≈64	6.39 µs	6.94 µs	-	-
	13	≈64	6.52 µs	7.04 µs	-	-
	14	≈64	6.63 µs	7.26 µs	-	-
	15	≈64	6.9 µs	7.45 µs	-	-
	16	≈64	7.57 µs	8.19 µs	-	-

$q=2^{64}-2^{32}+1$

Implementation Results: multiple NTT

$log_2 n$	RTX 3060 Ti		A100		RTX 4090	
	merge	4-step	merge	4-step	merge	4-step
12	51.7 µs	53.9 µs	27.1 µs	27.6 µs	13.3 µs	6.75 µs
13	95.6 µs	103.0 µs	43.0 µs	48.1 µs	21.6 µs	23.3 µs
14	185.9 µs	205.7 µs	80.8 µs	88.7 µs	39.6 µs	42.9 µs
15	362.1 µs	401.0 µs	163.8 µs	175.1 µs	75.9 µs	82.7 µs
16	750.4 µs	789.3 µs	339.9 µs	348.1 µs	164.2 µs	274.2 µs
17	2253.8 µs	2412.0 µs	746.6 µs	727.0 µs	827.2 µs	873.4 µs
18	4625.0 µs	4762.4 µs	1518.5 µs	1479.6 µs	1747.3 µs	1746.0 µs

$q=2^{64}-2^{32}+1$ **NTT count = 128**

carrier prime for ntt (2^{64}-2^{32}+1)

In HE applications, we use RNS arithmetic

$\log_2 n$	merge 1*	merge 2**		RNS factor
12	8.2 µs	7.24 µs	≈25	64/25≈2.5
13	9.1 µs	7.5 µs	≈25	64/25≈2.5
14	10.4 µs	9.51 µs	≈25	64/25≈2.5
15	13.9 µs	10.93 µs	≈24	64/24≈2.6
16	18.9 µs	13.8 µs	≈24	64/24≈2.6

* random prime

** carrier prime $P = 2^{64}-2^{32}+1$

$P > nq_i^2$

NTT for ZK proofs

- comparison with the fastest NTT at zprize competition
  - https://github.com/supranational/sppark/tree/main/ntt
- 256 bit prime, n = 2^{24}

A100 80 GB			RTX 4090		
t.w. (merge)	sparkk	speedup	t.w. (merge)	sparkk	speedup
7380.9 µs	7800.0 µs	%5	4478.1 µs	4858.7 µs	%7.8

BFV on GPU

GPU library can work with

- Seal
- Independent of SEAL
- Other operations included in the library:
  - Public & Secret Key Generation
  - Evaluation Keys Generation
  - Encoding & Decoding
  - Encryption & Decryption
  - Plain Addition & Plain Multiplication

Feature	CPU	GPU	
Model	AMD Ryzen7 3800X	RTX3060Ti	GTX1080
Threads	16	4864	2560
Frequency	4.20 GHz	1665 MHz	1733 MHz
RAM	32 GB (3600 MHz)	8 GB	8 GB

Operation	n	$\log_2 q$	GPU Library		SEAL	T
			RTX3060Ti	GTX1080	CPU	T_s
Addition	2^{12}	109	4 μs	4.6 μs	14 μs	3.5×
	2^{13}	218	5.1 μs	6.1 μs	58 μs	11.37×
	2^{14}	438	**12.3 μs**	19.4 μs	**233 μs**	**18.94**×
	2^{15}	881	44 μs	64.2 μs	778 μs	17.68×
Multiplication	2^{12}	109	86 μs	155.8 μs	3212 μs	37.3×
	2^{13}	218	202 μs	423.4 μs	11883 μs	58.8×
	2^{14}	438	**768 μs**	1856.1 μs	**48757 μs**	**63.4**×
	2^{15}	881	3757 μs	-	205295 μs	54.6×
Relinearization	2^{12}	109	39.51 μs	59.3 μs	625 μs	15.81×
	2^{13}	218	88.54 μs	143.4 μs	3100 μs	35.01×
	2^{14}	438	**376.61 μs**	825.3 μs	**18295 μs**	**48.57**×
	2^{15}	881	3150 μs	-	111736 μs	35.47×
Rotation	2^{12}	109	42.1 μs	59.4 μs	642 μs	15.24×
	2^{13}	218	103.3 μs	162.7 μs	3157 μs	30.56×
	2^{14}	438	**458.7 μs**	1067.2 μs	**18338 μs**	**39.97**×
	2^{15}	881	3464.5 μs	-	113437 μs	32.74×

enCRYPTON

Sabancı Üniversitesi

Homomorphic ML Inference I-DASH 2020 Competition

		SEAL		T.W		
n	$\log_2 q$	S.T.	M.T.	RTX 3060Ti	T	S
2^{13}	218	25.62 s	3.4 s	0.596 s	42.98×	5.7×
2^{14}	438	127.028 s	19.27 s	2.41 s	52.7×	8×

S.T.:Single Thread **M.T.**:Multi Thread.(16 threads) **T.W.**:This Work. T:The ratio of single-thread results over this work. S: The ratio of multi-thread results over this work.

Feature	CPU	GPU	
Model	AMD Ryzen7 3800X	RTX3060Ti	GTX1080
Threads	16	4864	2560
Frequency	4.20 GHz	1665 MHz	1733 MHz
RAM	32 GB (3600 MHz)	8 GB	8 GB

Conclusion

- GPUs are powerful devices
- They can be used for cryptographic acceleration
- Their capabilities sometimes overestimated
- More study needed to realize their true potentials

Bibliography

- [3] A. Sah Ozcan, C. Ayduman, E. R. Turkoglu, and E. Savas, "Homomorphic encryption on gpu," Cryptology ePrint Archive, Paper 2022/1222, 2022, https://eprint.iacr.org/2022/1222. [Online]. Available: https://eprint.iacr.org/2022/1222
- [8] K. Shivdikar, G. Jonatan, E. Mora, N. Livesay, R. Agrawal, A. Joshi, J. Abellan, J. Kim, and D. Kaeli, "Accelerating polynomial multiplication for homomorphic encryption on gpus," 2022. [Online]. Available: https://arxiv.org/abs/2209.01290

Thank You

