Perancangan dan Percepatan Algoritma Enkripsi Homomorfik Total Skema Cheon, Kim, Kim, Song (CKKS) dengan GPU

Daniel Mario Reynaldi
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132
13519031@std.stei.itb.ac.id

***Abstrak*—Enkripsi homomorfik adalah algoritma enkripsi yang memungkinkan proses komputasi pada data terenkripsi. Salah satu contoh algoritma enkripsi homomorfik adalah algoritma enkripsi CKKS yang dapat mengenkripsi bilangan kompleks. Algoritma enkripsi homomorfik memiliki tingkat keamanan yang sangat tinggi, tetapi tingkat penggunaan enkripsi homomorfik masih rendah karena waktu eksekusinya yang tinggi. Oleh karena itu, diperlukan upaya paralelisasi algoritma enkripsi CKKS untuk menurunkan waktu eksekusi algoritma enkripsi homomorfik. Paralelisasi dengan GPU dipilih karena ketersediaan GPU yang tinggi serta kemampuan komputasi paralelnya yang tinggi untuk algoritma bersifat single instruction multi data stream. Implementasi paralel dilakukan dengan platform CUDA pada GPU Nvidia, diimplementasikan juga CKKS secara serial dengan bahasa pemrograman C++ sebagai pembanding. Pengujian fungsionalitas dilakukan dengan mengenkripsi dan kemudian mendekripsi gambar, hasilnya, implementasi berhasil mengenkripsi gambar tanpa mengurangi kualitas gambar secara signifikan. Pengujian kinerja dilakukan dengan menghitung waktu eksekusi algoritma. Hasilnya, implementasi paralel memiliki waktu eksekusi yang lebih cepat dibanding implementasi serial.**

Kata kunci—Enkripsi Homomorfik, CKKS, percepatan, GPU

I. Pendahuluan

Enkripsi homomorfik adalah algoritma enkripsi yang memungkinkan komputasi dilakukan langsung pada ciphertext tanpa perlu dilakukan dekripsi kembali ke plaintext terlebih dulu (Morris, 2013). Salah satu contoh skema enkripsi homomorfik total yang populer adalah Cheon, Kim, Kim, Song atau yang sering disingkat CKKS (Cheon et al., 2016). Enkripsi homomorfik memungkinkan data selalu berada pada keadaan terenkripsi bahkan saat harus dilakukan operasi pada data tersebut.

Skema enkripsi homomorfik CKKS membutuhkan waktu eksekusi yang sangat lama untuk ukuran masukan yang besar, terutama pada tahap encoding, decoding, dan perkalian polinom. Akibatnya, meskipun memiliki tingkat keamanan yang tinggi, penggunaan enkripsi homomorfik masih sangat minim. Namun, karena sebagian besar operasi pada skema enkripsi homomorfik merupakan komputasi bertipe *single instruction multiple data stream* (SIMD), paralelisasi operasi-operasi skema enkripsi homomorfik dapat mengurangi lamanya waktu eksekusi.

Paralelisasi skema enkripsi homomorfik pada graphical processing unit dapat mengurangi waktu eksekusi operasi-operasi homomorfik dan akan meningkatkan penggunaan enkripsi homomorfik. Terlebih lagi, belum banyak implementasi paralel enkripsi homomorfik secara lengkap yang dilakukan pada platform CUDA.

II. Studi LIteratur

A. Enkripsi Homomorfik

Homomorfisma pada aljabar didefinisikan sebagai pemetaan antara dua struktur aljabar serupa yang mempertahankan operasi-operasi struktur aljabar tersebut (Yi et al., 2014). Jika terdapat struktur aljabar A, struktur serupa B, dan operasi berlaku bagi kedua struktur A dan B, maka sebuah homomorfisma f : A B adalah peta yang memiliki sifat

$$f(x) \bullet f(y) = f(x \bullet y) \quad (2.1)$$

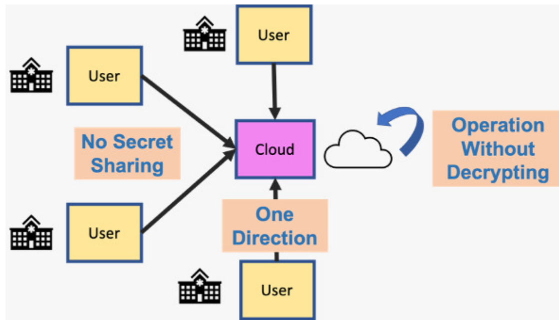
untuk setiap pasangan x,y yang merupakan elemen dari A. Mengacu pada definisi di atas, enkripsi homomorfik Ek, dengan k sebagai kunci, adalah sebuah fungsi enkripsi Ek : PC yang memiliki sifat sebagai berikut.

$$E_k(P \bullet P') = E_k(P) \bullet E_k(P') \quad (2.2)$$

Artinya, enkripsi homomorfik memungkinkan komputasi dilakukan langsung pada ciphertext tanpa perlu dilakukan dekripsi kembali ke plaintext terlebih dulu (Morris, 2013).

Sebagai contoh, pada lingkungan komputasi awan, pengguna layanan penyimpanan awan biasanya harus melakukan pertukaran kunci dengan penyedia jasa layanan penyimpanan awan agar data yang dienkripsi oleh pengguna dapat didekripsi sebelum dilakukan komputasi terhadap data. Mekanisme pertukaran kunci ini berpotensi menjadi titik lemah yang dapat dieksploitasi oleh pihak tidak berwenang. Selain itu, jika penyedia jasa layanan mengalami kebocoran data, pihak yang mendapatkan data pengguna dapat dengan mudah melihat data tersebut. Enkripsi homomorfik memungkinkan pengguna layanan melakukan enkripsi, mengirim data ke penyedia layanan, dan penyedia layanan

dapat melakukan komputasi pada cipherteks tanpa adanya proses pertukaran kunci.



Gambar 1.1 Implementasi Secure Cloud Computing dengan Enkripsi Homomorfik

B. Enkripsi Homomorphic SKEMA CKKS

Skema CKKS pertama kali dikemukakan oleh Jung Hee Cheon, Andrey Kim, Miran Kim, dan Yongsoo Song pada sebuah paper dengan judul "Homomorphic Encryption for Arithmetic of Approximate Numbers". Perbedaan skema CKKS dibanding algoritma FHE sebelumnya seperti BFV dan BGV adalah CKKS mendukung approximate arithmetic pada bilangan kompleks, berbeda dengan BFV dan BGV yang plainteksnya merupakan bilangan bulat (Cheon et al., 2016).

Secara garis besar, skema CKKS terdiri atas lima algoritma utama, yaitu pembangkitan kunci, enkripsi, dekripsi, penjumlahan dan perkalian homomorfik (Cheon, 2016). Tetapi karena pesan masukan skema CKKS adalah bilangan kompleks, pertama-tama harus dilakukan enkode terlebih dahulu dari pesan yang berupa vektor dengan elemen bilangan kompleks ke polinom dengan koefisien bilangan bulat, dan setelah proses dekripsi kita juga perlu melakukan dekode keluaran kembali ke bentuk vektor bilangan kompleks.

Skema CKKS memanfaatkan persoalan RLWE pada cincin, namun karena masukan skema CKKS adalah vektor dengan elemen berupa bilangan kompleks diperlukan sebuah metode untuk mentransformasikan vektor tersebut ke dalam bentuk plainteks yang berupa polinom . Proses enkode dapat digambarkan dengan persamaan berikut.

$$m(X) = \sigma^{-1}(\lfloor \Delta \pi^{-1}(z) \rceil_{\sigma(R)}) \quad (2.3)$$

Proses dekode jauh lebih sederhana dibandingkan enkode, polinom dievaluasi pada akar-akar cyclotomic polynomial yang digunakan, kemudian hasilnya dikalikan dengan . Proses dekripsi dapat digambarkan dengan persamaan berikut (Cheon et al., 2016).

$$z = \pi \circ \sigma(\Delta^{-1} m) \quad (2.4)$$

Proses pembangkitan kunci dimulai dengan memilih sebuah polinom sebagai kunci privat , kunci tersebut tidak disebarluaskan dan berguna untuk proses dekripsi cipherteks. Selanjutnya secara random dipilih dua buah polinom , kunci publik pk yang berguna untuk enkripsi plainteks didapatkan dengan persamaan berikut (Cheon et al., 2016).

$$pk = (-a \cdot s + e, a) \quad (2.5)$$

Enkripsi plainteks dengan kunci publik pk ke dalam bentuk cipherteks c digambarkan dengan persamaan sebagai berikut (Cheon et al., 2016).

$$c = \mu + pk = (\mu - a \cdot s + e, a) \quad (2.6)$$

Sebaliknya, dekripsi cipherteks c kembali ke plainteks dengan kunci privat sk digambarkan dengan persamaan sebagai berikut (Cheon et al., 2016).

$$\mu \approx c_0 + c_1 \cdot s \quad (2.7)$$

Skema CKKS merupakan skema homomorfik total, artinya skema tersebut mendukung operasi penjumlahan dan perkalian. Operasi penjumlahan pada CKKS dapat dilakukan dengan menjumlahkan dua atau lebih cipherteks, misalkan terdapat cipherteks dan , jika kedua cipherteks tersebut dijumlahkan maka akan didapatkan . Jika nilai penjumlahan tersebut didekripsi maka akan didapatkan hasil sebagai berikut.

$$(c_0 + c_0') + (c_1 + c_1') \cdot s = \mu + \mu' + 2e \approx \mu + \mu' \quad (2.8)$$

Operasi perkalian antar dua cipherteks dan akan menghasilkan persamaan berikut.

$$c_{mult} = (d_0, d_1, d_2) = (c_0 \cdot c_0', c_0 \cdot c_1' + c_0' \cdot c_1, c_1 \cdot c_1') \quad (2.9)$$

Perkalian antar cipherteks menghasilkan tiga polinomial, maka diperlukan metode khusus untuk dekripsi perkalian polinomial, dekripsi tersebut digambarkan dengan persamaan berikut.

$$D(c_{mult}, s) = d_0 + d_1 \cdot s + d_2 \cdot s^2 \approx \mu \cdot \mu' \quad (2.10)$$

C. Model Pemrograman CUDA

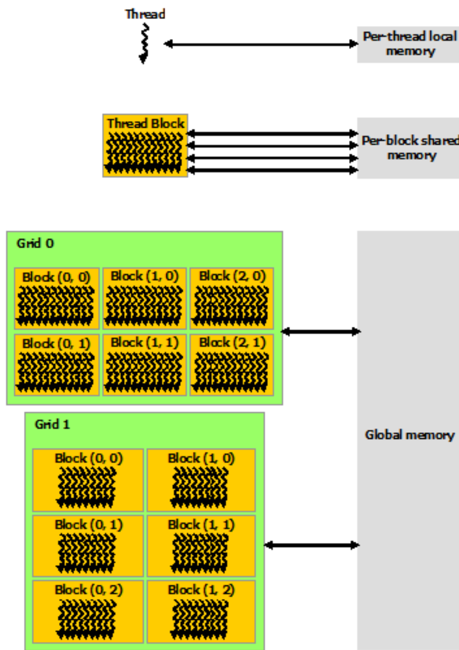
Pengembangan algoritma paralel dengan platform CUDA membutuhkan pemahaman akan beberapa konsep dasar yaitu kernel dan hirarki thread dan hirarki memori.

Kernel pada CUDA dapat dianalogikan sebagai fungsi pada bahasa pemrograman pada umumnya. Namun, berbeda dengan fungsi yang hanya dieksekusi sekali oleh CPU, kernel dieksekusi sebanyak N kali secara paralel oleh N thread pada GPU. Kernel ditandai dengan specifier __global__, saat kernel akan dipanggil maka harus dideklarasikan jumlah thread yang akan digunakan untuk mengeksekusi fungsi tersebut dengan sintaks <<…>>.

Thread pada CUDA dapat diidentifikasi dengan vektor tiga dimensi. Saat pemanggilan kernel, thread yang digunakan dapat didefinisikan dalam vektor satu, dua, atau tiga dimensi yang disebut block. Tiap block memiliki maksimal 1024 thread, dan setiap pemanggilan kernel dapat dieksekusi oleh lebih dari satu block (NVIDIA, 2022).

Sebuah thread pada CUDA dapat mengakses data dari beberapa lokasi memori selama melakukan eksekusi kernel. Semua thread memiliki sebuah memori lokal yang hanya

dapat diakses oleh thread tersebut, setiap block memiliki sebuah shared memory yang dapat diakses oleh seluruh thread pada sebuah block, dan terdapat sebuah global memory yang dapat diakses oleh semua thread dari semua block.



Gambar 1.2 Hirarki memori CUDA

(NVIDIA CUDA Toolkit, 2022)

III. Rancangan Solusi

Setiap tahap enkripsi CKKS seperti yang telah dijelaskan pada studi literatur diimplementasikan dengan bahasa pemrograman C++ secara serial pada CPU dan paralel pada GPU dengan platform CUDA. Paralelisasi pada tahap ini bersifat naive, artinya, strategi paralelisasi hanya membagi setiap operasi aritmatika ke masing-masing thread CUDA pada GPU tanpa mekanisme optimasi.

A. Rancangan Solusi Serial

Tahap encoding terdiri dari tahap pi inverse, *scale, discretization,* dan *sigma inverse*. Berikut merupakan rancangan algoritma *encoding* dalam notasi pseudocode.

Algorithm 2: Pi Inverse

Input: array of complex number z
Output: array of complex number z'
$n \leftarrow length(z)$;
$z'[n] \leftarrow \{\}$;
for j=*0 to N-1* **do**
  $z'[j] \leftarrow z[j]$;
  $z'[2j-(i+1)] \leftarrow conj(z[j])$;
end

Gambar 3.1 Rancangan Algoritma *pi inverse*

Algorithm 3: scale

Input: array of complex number z and scale Δ
Output: array of complex number z'
$n \leftarrow length(z)$;
$z'[n] \leftarrow \{\}$;
for j=*0 to N-1* **do**
  $z'[j] \leftarrow \Delta \cdot z[j]$;
end

Gambar 3.2 Rancangan Algoritma *scale*

Algorithm 5: Sigma Inverse

Input: array bilangan real zR
Output: Polynomial pt
for j=*0 to N-1* **do**
  $zR[j] \leftarrow round(zR[j])$;
end
$pt \leftarrow setCoeff(zR)$;

Gambar 3.3 Rancangan Algoritma *sigma inverse*

Tahap decoding terdiri dari algoritma sigma yaitu evaluasi polinom serta pembagian koefisien polinom dengan sebuah bilangan bulat, tahap tersebut secara lebih detail telah dibahas pada bagian studi literatur. Berikut merupakan rancangan algoritma *decoding* dalam notasi pseudocode.

Algorithm 6: sigma

Input: Polynomial P, scale Δ and vandermonde matrix V
Output: array of complex number z
$M \leftarrow length(P)$;
$N \leftarrow \frac{M}{2}$;
$z[n] \leftarrow \{\}$;
for j=*0 to N-1* **do**
  for k=*0 to M-1* **do**
    $z[j] \leftarrow z[j] + (\frac{1}{\Delta} \cdot P[k] \cdot V[j][k])$;
  end
end

Gambar 3.4 Rancangan Algoritma *sigma*

Tahap enkripsi terdiri dari tahap pembangkitan kunci publik, kunci privat, dan kunci evaluasi, serta proses enkripsi. Berikut merupakan rancangan algoritma enkripsi dalam notasi pseudocode.

Algorithm 12: Encrypt

Input: Public Key (b, a) and Plaintext μ
Output: Ciphertext $ct = (c_0, c_1)$
$c_0 \leftarrow \mu + b$;
$c_1 \leftarrow a$;

Gambar 3.5 Rancangan Algoritma Enkripsi Serial

Tahap dekripsi terdiri dari pengembalian plainteks dari cipherteks dengan masukan berupa kunci privat s dan cipherteks (c0, c1). Berikut merupakan rancangan algoritma dekripsi.

Algorithm 13: Decrypt

Input: secret key s, Ciphertext $ct = (c_0, c_1)$
Output: Plaintext μ
$\mu \leftarrow c_0 + c_1 \cdot s$;

Gambar 3.6 Rancangan Algoritma Dekripsi Serial

Evaluasi homomorfik pada skema CKKS terdiri dari operasi penjumlahan dan perkalian cipherteks. Berikut merupakan merupakan penjumlahan dua buah cipherteks, dimana setiap koefisien cipherteks saling dijumlahkan satu sama lain.

Algorithm 14: Ciphertext Add

Input: Ciphertext $ct = (c_0, c_1)$ and Ciphertext $ct' = (c'_0, c'_1)$
Output: Ciphertext $ct_{add} = c_0, c_1$
$ct_{add} \leftarrow (c_0 + c'_0) + (c_1 + c'_1)$;

Gambar 3.7 Rancangan Algoritma Penjumlahan Cipherteks

Perkalian cipherteks menerima masukan berupa dua buah cipherteks dan menghasilkan triplet polinom. Selanjutnya, dilakukan proses relin yang mereduksi triplet kembali ke bentuk pasangan polinom, tujuannya agar ukuran cipherteks tidak semakin membesar setiap dilakukan perkalian. Algoritma relin menerima triplet hasil perkalian cipherteks dan evaluation key yang telah dibangkitkan pada tahap enkripsi, dan menghasilkan pasangan polinom. Terakhir, dilakukan rescaling yang berfungsi mengurangi error yang diakibatkan saat perkalian cipherteks.

Algorithm 15: Ciphertext Mult

Input: Ciphertext $ct = (c_0, c_1)$ and Ciphertext $ct' = (c'_0, c'_1)$
Output: Polynomial Triplet $ct_{mult} = (c_0, c_1, c_2)$
$ct_{mult} \leftarrow (c_0 * c'_0), (c_0 * c'_1 + c_1 * c'_0), (c_1 + c'_1)$;

Gambar 3.8 Rancangan Algoritma Perkalian Cipherteks

Algorithm 16: Relin

Input: Polynomial Triplet $ct_{mult} = (c_0, c_1, c_2)$, Big Integer P, Eval key evk
Output: Ciphertext $ct' = (c'_0, c'_1)$
$ct' \leftarrow (c_0, c_1) + (c_2 \cdot evk \cdot P^{-1})$;

Gambar 3.9 Rancangan Algoritma *relin*

Algorithm 17: Rescaling

Input: Ciphertext $ct = (c_0, c_1)$, constant q_l and q_{l-1}
Output: Ciphertext $ct' = (c'_0, c'_1)$
$ct' \leftarrow \frac{q_{l-1}}{q_l} \cdot (c0, c1)$;

Gambar 3.10 Rancangan Algoritma *rescaling*

Tahap NTT berguna mengubah polinom ke bentuk representasi point-value, dan sebaliknya iNTT mengubah representasi kembali ke bentuk polinom. Tujuan dilakukannya tahap ini adalah untuk mempercepat perkalian polinom yang sering terjadi pada skema CKKS.

Algorithm 18: NTT

Input: array of integer P, nth root of unity ω, boolean $isInverse$, modulo M
Output: array of integer ntt
$Nbit \leftarrow bitLen(P)$;
$N \leftarrow length(P)$;
$ntt[N] \leftarrow \{\}$;
if $isInverse$ **then**
  $\omega \leftarrow \omega^{-1}$
end
for j=0 *to* $Nbit$ **do**
  $p1[N/2] \leftarrow \{\}$;
  $p2[N/2] \leftarrow \{\}$;
  for k=0 *to* $N/2$ **do**
    $shift \leftarrow Nbit - j - 1$;
    $P \leftarrow (k >> shift) << shift$;
    $wP \leftarrow \omega^P \% M$;
    $odd \leftarrow P[2 * k + 1] * wP$;
    $even \leftarrow P[2 * k]$;
    $p1[k] \leftarrow (even + odd)\%M$;
    $p2[k] \leftarrow (even - odd)\%M$;
  end
  $P \leftarrow concat(p1, p2)$;
end
$ntt \leftarrow P$;

Gambar 3.11 Rancangan Algoritma NTT Serial

B. Rancangan Solusi Paralel

Rancangan tahap encoding dan decoding secara paralel dengan platform CUDA, menggunakan prinsip yang serupa dengan algoritma serial. Tetapi pada algoritma paralel, tidak terdapat tahap iterasi karena setiap tahap algoritma dieksekusi secara bersamaan oleh thread yang berbeda pada GPU.

Algorithm 19: Parallel Encoding

Input: array of complex number z, scale Δ
Output: Plaintext Polynomial P
$block \leftarrow length(z)/1024$;
$threadPerBlock \leftarrow 1024$;
$z' \leftarrow expScale <<< block, threadPerBlock >>> (z, \Delta)$;
$coords \leftarrow ComputeCoordinate <<< block, threadPerBlock >>> (z')$;
$P \leftarrow RoundCoordinate <<< block, threadPerBlock >>> (coords)$;

Gambar 3.12 Rancangan Algoritma *Encoding* Paralel

Algorithm 23: Parallel Decoding

Input: Polynomial P, vandermonde matrix V, scale Δ
Output: array of complex number z
$block \leftarrow length(z)/1024$;
$threadPerBlock \leftarrow 1024$;
$P' \leftarrow unscale <<< block, threadPerBlock >>> (P, \Delta)a$;
$z \leftarrow ParallelSigma <<< block, threadPerBlock >>> (P', V)$;

Gambar 3.13 Rancangan Algoritma *Decoding* Paralel

Prinsip algoritma enkripsi dan dekripsi serupa dengan rancangan serial tetapi pada algoritma enkripsi paralel setiap koefisien polinom pada plainteks akan dijumlahkan dengan satu koefisien pada kunci publik oleh sebuah thread, dan total

thread yang sejumlah dengan derajat polinom plainteks dan kunci publik.

Algorithm 26: Parallel Encryption
Input: Polynomial Plaintext P, Public Key (b, a)
Output: Ciphertext $ct(c_0, c_1)$
$idx \leftarrow threadIdx.x + blockDim.x * blockIdx.x;$
$c_0[idx] \leftarrow b[idx] + P[idx];$
$c_1[idx] \leftarrow a[idx];$

Gambar 3.14 Rancangan Algoritma Enkripsi Paralel

Algorithm 27: Parallel Decryption
Input: Ciphertext $ct(c_0, c_1)$, Secret Key s
Output: Polynomial Plaintext P
$idx \leftarrow threadIdx.x + blockDim.x * blockIdx.x;$
$P[idx] \leftarrow c_0[idx] + c_1[idx] * s[idx];$

Gambar 3.15 Rancangan Algoritma Dekripsi Paralel

Prinsip algoritma enkripsi dan dekripsi serupa dengan rancangan serial tetapi pada algoritma enkripsi paralel setiap koefisien polinom pada plainteks akan dijumlahkan dengan satu koefisien pada kunci publik oleh sebuah thread, dan total thread yang sejumlah dengan derajat polinom plainteks dan kunci publik.

Algorithm 28: Parallel Ciphertext Addition
Input: Ciphertext $ct(c_0, c_1)$, Ciphertext $ct'(c'_0, c'_1)$
Output: Ciphertext $ct_{add}(c_{0add}, c_{1add})$
$idx \leftarrow threadIdx.x + blockDim.x * blockIdx.x;$
$c_{0add}[idx] \leftarrow c_0[idx] + c'_0[idx];$
$c_{1add}[idx] \leftarrow c_1[idx] + c'_1[idx];$

Gambar 3.16 Rancangan Algoritma Penjumlahan Cipherteks Paralel

Operasi perkalian, relinearisasi dan rescaling tetap menggunakan alur seperti implementasi serial tetapi pada algoritma paralel, operasi perkalian polinom bersifat paralel.

Algorithm 29: Parallel Polynomial Multiplication
Input: Polynomial $P1$, Polynomial $P2$
Output: Polynomial $P3$
$idx \leftarrow threadIdx.x + blockDim.x * blockIdx.x;$
$P3[idx] \leftarrow 0;$
for *j=0 to P1.degree* **do**
  for *k=0 to P2.degree* **do**
    if *j+k == idx* **then**
      $P3[idx] \leftarrow P3[idx] + P1[j] * P2[k]$
    end
  end
end

Gambar 3.17 Rancangan Algoritma Perkalian Polinom Paralel

Tahap NTT dan iNTT memiliki prinsip kerja yang serupa dengan algoritma serial yang sudah dibahas pada III.3.1.4. Tetapi, algoritma NTT dan iNTT paralel dieksekusi oleh n/2 thread pada GPU, dimana n merupakan derajat polinom yang akan diubah ke bentuk representasi point value.

Algorithm 30: Parallel NTT
Input: array of integer P, nth root of unity ω, boolean $isInverse$, modulo M
Output: array of integer ntt
$Nbit \leftarrow bitLen(P);$
$N \leftarrow length(P);$
$ntt[N] \leftarrow \{\};$
if $isInverse$ **then**
  $\omega \leftarrow \omega^{-1}$
end
$block \leftarrow length(z)/1024;$
$threadPerBlock \leftarrow 1024;$
for *j=0 to Nbit* **do**
  $p1[N/2] \leftarrow \{\};$
  $p2[N/2] \leftarrow \{\};$
  $computeNTT <<< block, threadPerBlock >>> (p1, p2, N, j, \Delta, nBit, P);$
  $P \leftarrow concat(p1, p2);$
end
$ntt \leftarrow P;$

Gambar 3.18 Rancangan Algoritma NTT Paralel

Algorithm 31: Compute NTT
Input: array of integer $p1$, nth root of unity ω, integer N, j, $Nbit$
Output: array of integer $p1$, $p2$
$idx \leftarrow threadIdx.x + blockDim.x * blockIdx.x;$
$shift \leftarrow Nbit - i - 1;$
$P \leftarrow (idx >> shift) << shift;$
$wP \leftarrow \omega^P \% M;$
$odd \leftarrow P[2 * idx + 1] * wP;$
$even \leftarrow P[2 * idx];$
$p1[idx] \leftarrow (even + odd)\%M;$
$p2[idx] \leftarrow (even - odd)\%M;$

Gambar 3.19 Rancangan Algoritma *Compute NTT*

IV. PENGUJIAN

A. Lingkungan Pengujian

Implementasi dan Pengujian dilakukan pada lingkungan perangkat dengan spesifikasi sebagai berikut.

Perangkat Keras:

- Processor: Intel Core I7 9700K
- Memory: 16 GB 3200 MHz
- Storage: SSD 480 GB
- GPU: Nvidia Geforce GTX 1660

Perangkat Lunak:

- Sistem Operasi: Ubuntu 20.04
- Compiler Kode Serial: G++ version 9.4.0
- Compiler CUDA: NVCC versi 10.1
- IDE: Visual Studio Code

B. Pengujian Fungsionalitas

Pengujian fungsionalitas dilakukan secara kualitatif dan kuantitatif. Secara kualitatif gambar masukan dibandingkan dengan gambar hasil dekripsi. Gambar IV-23 merupakan perbandingan gambar masukan yang kemudian dienkripsi

dengan implementasi CKKS dan didekripsi, serta gambar keluaran hasil dekripsi. Dapat dilihat bahwa kedua gambar tersebut terlihat serupa, dan tidak terdapat perbedaan yang dapat dilihat dengan mata.

Pengujian kuantitatif dilakukan dengan mengukur nilai PSNR dari Gambar IV-23 dan Gambar IV-24. Menurut perhitungan, nilai PSNR yang didapatkan adalah 102.273 dB. Nilai tersebut dinilai memenuhi kriteria enkripsi gambar yang bagus karena lebih besar dari 30 dB.

Gambar 4.1 Perbandingan Gambar Masukan dan Gambar

Hasil Dekripsi

C. Pengujian Kinerja

Pengujian dilakukan untuk mengukur waktu eksekusi tahap-tahap pada algoritma enkripsi homomorfik CKKS secara paralel dan serial. Pengujian juga mengukur percepatan waktu eksekusi serial dibanding paralel. Tabel I merupakan hasil pengujian kinerja pada CKKS dengan masukan berjumlah 8192.

TABLE I. Hasil Pengujian Kinerja Implementasi CKKS

Operasi	***Waktu Serial***	***Waktu Paralel***	***Speedup***
Encoding	83284134.6	23.6	3528988x
Decoding	25544100.2	7.4	3451905x
Pembangkitan Kunci Privat	297	203	1.4x
Pembangkitan Kunci Publik	179472.6	1266	141x
Pembangkitan Kunci Evaluasi	90145.8	633	142x
Enkripsi	2147.4	184	11x
Dekripsi	89253.6	18.6	4798x
Penjumlahan Cipherteks	535.8	19.8	27x
Perkalian Cipherteks	1316268.2	413	3187x
Perkalian Cipherteks dengan NTT	3086285.8	108481.8	28x

V. Kesimpulan

Enkripsi homomorfik skema CKKS berhasil diimplementasikan secara paralel dengan platform CUDA dan secara serial dengan bahasa C++. Kedua implementasi berhasil mengenkripsi gambar tanpa mengurangi kualitas gambar. Perkalian polinom dengan *number theoretic transform* juga berhasil diimplementasikan secara serial dengan bahasa C++ dan secara paralel dengan platform CUDA. Pengukuran kinerja implementasi CKKS menunjukan bahwa implementasi paralel lebih cepat dibandingkan implementasi serial, artinya paralelisasi berhasil mengurangi waktu eksekusi enkripsi homomorfik CKKS.

VI. Ucapan Terima Kasih

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa, karena atas rahmat-Nya penulis dapat menyelesaikan penulisan *paper* ini. Penulis berterima kasih kepada Dr. Ir. Rinaldi Munir, M.T. dan Dr. Eng. Infall Syafalni, S.T., M.Sc. selaku pembimbing penulis yang telah telah memberikan bimbingan yang sangat membantu dan menginspirasi pengerjaan *paper* ini. Penulis juga berterima kasih kepada seluruh teman seangkatan di IF'19 yang telah menemani dan membantu penulis selama masa perkuliahan di ITB.

References

[1] Bull, D. R., & Zhang, F. (2021). Intelligent Image and Video Compression Communicating Pictures. Elsevier.

[2] Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. Advances in Cryptology – ASIACRYPT 2017, 409–437.

[3] Cormen, T. H., & Leiserson, C. E. (2009). Introduction to algorithms, 3rd Edition. The MIT Press.

[4] Delfs, H., & Knebl, H. (2002). Introduction to cryptography: Principles and applications. Springer.

[5] Fan, J. & Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption.. IACR Cryptology ePrint Archive, 2012, 144.

[6] Introduction. CUDA C++ Programming Guide. (2022, December 8). Retrieved December 13, 2022, from https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[7] Lang, S. (2008). Algebra (3rd ed.). Springer.

[8] Pacheco, P. S. (2013). An introduction to parallel programming. Elsevier Science & Technology.

[9] Paillier, P. (n.d.). Public-key cryptosystems based on composite degree residuosity classes. Advances in Cryptology — EUROCRYPT '99, 223–238. https://doi.org/10.1007/3-540-48910-x_16

[10] Sanders, J., & Kandrot, E. (2010). Cuda C by example: An introduction to general-purpose Gpu programming. Addison-Wesley.

[11] Trobec, R., Slivnik, B., Bulić, P., & Robič, B. (2018). Introduction to parallel computing: From algorithms to programming on state-of-the-art platforms. Springer.