Implementasi Paralel Enkripsi Homomorfik Total Kunci Jamak Skema Chen, Dai, Kim, Song (CDKS) pada GPU dengan CUDA

Ilham Prasetyo Wibowo
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
13520013@std.stei.itb.ac.id

Rinaldi Munir
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
rinaldi@staff.itb.ac.id

Infall Syafalni
Program Studi Teknik Elektro
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
infall@staff.itb.ac.id

***Abstract*— Enkripsi homomorfik memungkinkan komputasi pada data terenkripsi. Skema Cheon-Kim-Kim-Song (CKKS) mendukung penjumlahan dan perkalian pada teks terenkripsi tetapi hanya dengan kunci yang sama. Skema Chen-Dai-Kim-Song (CDKS) memperluas CKKS dengan memungkinkan operasi pada teks terenkripsi dengan kunci berbeda, yang dikenal sebagai Multi-Key Fully Homomorphic Encryption (MK-FHE). Meskipun memiliki potensi besar, CDKS memerlukan sumber daya komputasi yang tinggi, yang mengakibatkan waktu eksekusi yang lama jika dijalankan secara serial. Paralelisasi menggunakan CUDA pada GPU NVIDIA dapat mempercepat CDKS. Hasil menunjukkan bahwa implementasi paralel mencapai kecepatan eksekusi hingga 50 kali lebih cepat dibandingkan dengan implementasi serial yang ada.**

Keywords—Homomorfik, CDKS, GPU, CUDA

I. PENDAHULUAN

Enkripsi homomorfik adalah bentuk enkripsi yang memungkinkan jenis-jenis komputasi tertentu dilakukan pada ciphertext, menghasilkan hasil terenkripsi yang, ketika didekripsi, sesuai dengan operasi yang dilakukan pada plaintext [1]. Salah satu skema enkripsi homomorfik total yang terkenal adalah skema HEAAN [2]. Skema ini, yang sering disebut sebagai CKKS, memungkinkan komputasi pada data terenkripsi, menjaga kerahasiaan data sepanjang proses tersebut.

Selain itu, CKKS hanya mendukung operasi homomorfik pada ciphertext yang dienkripsi dengan kunci yang sama. Pada tahun 2019, Chen et al. mengusulkan varian dari skema CKKS yang mendukung operasi homomorfik pada ciphertext yang dienkripsi dengan kunci berbeda [3]. Varian banyak kunci ini, yang disebut CDKS, memungkinkan komputasi pada data yang dienkripsi dengan kunci berbeda, menjadikannya cocok untuk skenario komputasi awan di mana beberapa pihak ingin melakukan komputasi pada data mereka sambil menjaga kerahasiaan.

Meskipun memiliki potensi besar, skema CDKS menuntut sumber daya komputasi yang lebih tinggi dibandingkan dengan skema CKKS standar. Penggunaan GPU (Graphical Processing Units) menghadirkan solusi yang menjanjikan untuk meningkatkan kinerja operasi CDKS. GPU dapat mempercepat permintaan komputasi tinggi dari skema CDKS, membuatnya lebih layak untuk aplikasi praktis.

CUDA adalah platform komputasi paralel dan model pemrograman yang dikembangkan oleh NVIDIA untuk komputasi umum pada GPU [4]. Implementasi CDKS pada GPU yang mendukung CUDA dapat secara signifikan meningkatkan kinerja operasi enkripsi homomorfik. Seiring dengan meningkatnya jumlah pengguna yang menggunakan skema ini, ukuran ciphertext bertumbuh secara linear, dan operasi homomorfik dapat diparalelisasikan untuk mencapai kinerja yang lebih baik.

Penelitian sebelumnya telah menunjukkan manfaat dari paralelisasi skema enkripsi homomorfik pada GPU. Ozcan et al. (2022) mengembangkan pustaka untuk menjalankan skema BFV pada GPU [6]. Reynaldi (2023) mengimplementasikan skema CKKS pada platform CUDA secara paralel secara naif, mencapai percepatan yang signifikan [5]. Yang et al. (2023) memparalelkan skema CKKS dan BGV pada GPU dengan beberapa optimasi [3]. Namun, studi-studi ini belum menangani operasi homomorfik menggunakan banyak kunci, yang menjadi fokus dari skema CDKS.

II. STUDI LITERATUR

A. Enkripsi Homomorfik Total

Skema enkripsi homomorfik total dapat dilihat sebagai sebuah homomorfisma cincin yang mempertahankan struktur aljabar cincin pada data yang telah dienkripsi (Yi dkk., 2014). Dalam konteks ini, cincin mengacu pada struktur aljabar abstrak yang memiliki dua operasi, yaitu penjumlahan dan perkalian, yang memungkinkan untuk menjalankan operasi penjumlahan dan perkalian pada data yang dienkripsi dengan menjaga integritas struktur aljabar cincin.

Secara lebih rinci, jika terdapat dua cincin yang mewakili ruang plaintext dan ruang ciphertext, misalnya $(P, \oplus_p, \otimes_p)$ dan $(C, \oplus_c, \otimes_c)$, maka skema enkripsi homomorfik total akan mendefinisikan suatu fungsi E, yang memetakan elemen-elemen dari cincin P ke elemen-elemen dalam cincin C. Untuk setiap a,b ∈ P dan kunci k, fungsi E ini harus memenuhi persamaan berikut:

$$E_k(a) \oplus_c E_k(b) = E_k(a \oplus_p b) \quad (2.1)$$

$$E_k(a) \otimes_c E_k(b) = E_k(a \otimes_p b) \quad (2.2)$$

B. Skema Enkripsi Cheon-Kim-Kim-Song (CKKS)

Skema Enkripsi Cheon-Kim-Kim-Song (CKKS) adalah sebuah kontribusi penting dalam dunia kriptografi, terutama dalam konteks enkripsi homomorfik total. CKKS memungkinkan komputasi pada data yang terkait dengan bilangan kompleks atau bilangan real aproksimatif. Penemuan

ini tercantum dalam publikasi ilmiah yang berjudul "Homomorphic Encryption for Arithmetic of Approximate Numbers" yang ditulis oleh Jung Hee Cheon, Andrey Kim, Miran Kim, dan Yongsoo Song.

Untuk memahami CKKS dengan lebih mendalam, perlu dijelaskan bahwa skema ini beroperasi di dalam ruang plaintext yang terbentuk oleh suatu cincin matematika, yaitu $R = \mathbb{Z}[X]/\ (X^N + 1)$. Di dalam cincin ini, Z adalah himpunan bilangan bulat, X adalah variabel polinomial, dan $(X^N + 1)$ adalah modulus polinomial yang digunakan untuk merepresentasikan data. Namun, dalam dunia nyata, data seringkali memiliki sifat yang kontinu, seperti bilangan kompleks. Oleh karena itu, CKKS mengembangkan suatu mekanisme encoding dan decoding yang memungkinkan transformasi vektor data kompleks $z \in C^{N/2}$ menjadi bentuk plaintext $m(X) \in \mathbb{Z}[X]/\ (X^N + 1)$.

C. Skema Enkripsi Chen-Dai-Kim-Song (CDKS)

Skema Enkripsi Cheon-Kim-Kim-Song (CKKS) adalah sebuah skema enkripsi homomorfik yang mampu mendukung operasi homomorfik pada dua ciphertext yang dienkripsi dengan satu kunci rahasia (secret key) yang sama. Sehinga operasi homomorfik hanya dapat dijalankan pada dua ciphertext yang memiliki asal kunci rahasia yang serupa. Namun, pada tahun 2019, para peneliti, yaitu Hao Chen, Wei Dai, Miran Kim, dan Yongsoo Song, memperkenalkan variasi baru dari skema CKKS yang mengatasi pembatasan ini. Variasi ini dikenal sebagai skema Chen-Dai-Kim-Song (CDKS) dan dijelaskan dalam sebuah publikasi ilmiah berjudul "Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference."

Skema CDKS memungkinkan operasi homomorfik pada dua ciphertext yang dienkripsi dengan kunci-kunci rahasia yang berbeda. Dengan kata lain, skema ini mendukung komputasi yang aman pada data terenkripsi menggunakan kunci rahasia yang berbeda. Skema ini sangat berguna dalam situasi ketika berbagai pihak atau entitas memiliki kunci rahasia masing-masing dan ingin melakukan operasi homomorfik terhadap data terenkripsi secara bersamaan tanpa perlu berbagi kunci rahasia. Operasi encoding dan decoding pada variasi CKKS ini sama seperti CKKS single-key.

1) Pembangkitan Kunci

Pembangkitan kunci pada skema ini hampir sama dengan skema CKKS. Pembangkitan kunci dimulai dengan membangkitkan kunci rahasia *s* dari sebuah distribusi kunci tertentu yang dipilih. Dilanjutkan dengan melakukan *sampling* a, e secara random dan mengeset kunci publik sebagai $pk \leftarrow (b, a)$ yang dalam hal ini $b \leftarrow -a.s + e$. Bentuk kunci evaluasi pada skema ini berbeda, yaitu sebuah vektor $D = [d_0|d_1|d_2]$. Pembangkitan kunci evaluasi dimulai dengan melakukan sampel terhadap sebuah variabel random *r,* d_1 , dan *error* e_1 , e_2 dilanjutkan dengan mengeset $d_0 = -s\,.\,d_1 + e_1 + r\,.\,g$ dan $d_2 = r\,.\,a + e_2 + m\,.\,g$. Sebuah vektor g dalam pembangkitan kunci evaluasi ini adalah sebuah *gadget vector. Gadget vector* merupakan sebuah vektor yang menjadi basis saat proses *gadget decomposition. Gadget decomposition* adalah sebuah fungsi yang dinotasikan dengan g^{-1} yang memetakan dari cincin R_q menjadi cincin R^d . Sehingga untuk elemen $a \in R_q$ dapat dipetakan menjadi sebuah vektor $u \in R^d$ yang memenuhi persamaan berikut [3].

$$a = \sum_{i=0}^{d-1} g_i\,.\,u_i \tag{2.3}$$

2) Enkripsi dan Dekripsi

Proses enkripsi pada skema ini sama dengan proses enkripsi pada skema CKKS. Proses enkripsi dimulai dengan melakukan *sampling* pada v, e_0, e_1 kemudian mengeluarkan hasil enkripsi sebagai berikut [3].

$$E(m) = v\,.\,pk + (m + e_0, e_1) \in R_{qL}^2 \tag{2.4}$$

Proses dekripsi pada skema kunci multi ini terdapat dua cara. Cara yang pertama adalah dekripsi dengan semua kunci rahasia pengguna diketahui. Maka untuk sebuah *ciphertext* $ct = (c_0, c_1, \dots, c_k) \in R_{ql}^{k+1}$ yang berkorespondensi dengan k pengguna, serta sebuah kumpulan kunci rahasia $sk = (1, s_1, \dots, s_k)$ hasil dekripsi dari *ciphertext* adalah sebagai berikut [3].

$$m = \langle ct, sk \rangle \tag{2.5}$$

Dengan kata lain, *plaintext* diperoleh dengan melakukan operasi *dot product* pada *ciphertext* dan *secret key.* Cara lain untuk dekripsi sebuah *ciphertext* pada skema ini adalah dengan dekripsi terdistribusi. Artinya, seorang pengguna dengan kunci *s* dapat memperoleh hasil dekripsi parsialnya terhadap *plaintext.* Namun, untuk memperoleh nilai sebenarnya diperlukan operasi penggabungan hasil dekripsi parsial. Untuk melakukan dekripsi parsial sebuah *ciphertext* c_i dengan sebuah kunci rahasia s_i , sampel sebuah *error* secara acak *e,* dan keluarkan hasil dekripsi parsial $m_i = c_i\,.\,s_i + e$. Gabungan dari hasil dekripsi parsial memenuhi persamaan berikut [3].

$$m = c_0 + \sum_{i=1}^{k} m_i \tag{2.6}$$

3) Operasi Homomorfik

Sama halnya seperti operasi homomorfik pada skema CKKS, skema ini juga mendukung operasi penjumlahan dan perkalian dilakukan pada *ciphertext* seolah-olah operasi tersebut dilakukan pada *plaintext.* Operasi penjumlahan pada skema ini cukup sederhana dengan menjumlahkan kedua *ciphertext* secara langsung dan memperoleh *ciphertext* dengan ukuran lebih besar. Untuk sebuah *ciphertext* ct dan *ciphertext* lain ct' , penjumlahan kedua *ciphertext* mengembalikan $ct + ct'$. Sebagai contoh, untuk *ciphertext* $ct = (c_0, c_1)$ yang dienkripsi menggunakan kunci rahasia s_1 dan *ciphertext* $ct' = (c_0', c_1')$ yang dienkripsi menggunakan kunci s_2 hasil operasi penjumlahan kedua *ciphertext* tersebut adalah $ct_{add} = (c_0 + c_0', c_1, c_1')$ yang bisa didekripsi dengan kunci $s = (1, s_1, s_2)$.

Operasi perkalian dalam skema Chen-Dai-Kim-Song (CDKS) melibatkan serangkaian langkah yang serupa dengan operasi dalam skema Cheon-Kim-Kim-Song (CKKS), yaitu proses *tensor product*, relinearisasi, dan *rescale*. Selama proses tensor product, ukuran ciphertext awal, yang sebelumnya sebesar k + 1, akan berkembang menjadi ciphertext baru dengan ukuran $(k + 1)^2$, sehingga diperlukan pemrosesan tambahan saat melakukan dekripsi dengan kunci rahasia yang dikuadratkan. Cheon dkk. menyajikan sebuah algoritma relinearisasi yang bertujuan untuk mengembalikan ciphertext tersebut ke ukuran asalnya, dengan langkah-langkah yang terdapat pada Gambar II.1.

Relinearization Method

Input: $\overline{\mathsf{ct}} = (c_{i,j})_{0\le i,j\le k}, \{(\mathbf{D}_i = [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}|\mathbf{d}_{i,2}], \mathbf{b}_i)\}_{1\le i\le k}$.
Output: $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le k} \in R_q^{k+1}$.
1: $c'_0 \leftarrow c_{0,0}$
2: **for** $1 \le i \le k$ **do**
3: $\quad c'_i \leftarrow c_{0,i} + c_{i,0} \pmod q$
4: **end for**
5: **for** $1 \le i, j \le k$ **do**
6: $\quad c'_{i,j} \leftarrow \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{b}_j \rangle \pmod q$
7: $\quad (c'_0, c'_i) \leftarrow (c'_0, c'_i) + \mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}] \pmod q$
8: $\quad c'_j \leftarrow c'_j + \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \pmod q$
9: **end for**

Gambar II.1. Algoritma Relinearisasi [3]

Langkah terakhir pada operasi perkalian adalah melakukan prosedur *rescaling* dengan persamaan berikut (Cheon dkk., 2019).

$$c'_i = \lfloor p^{-1} c_i \rceil, i = 0, 1, 2, \dots, k \qquad (2.7)$$

D. *Pemrograman Paralel*

Peningkatan kinerja prosesor tunggal didorong oleh kepadatan transistor yang semakin tinggi, tetapi peningkatan kecepatan transistor juga meningkatkan konsumsi daya dan panas. Pada dekade pertama abad ke-21, sirkuit berpendingin udara mencapai batasan kemampuannya untuk menghilangkan panas [7]. Seiring dengan batasan panas, industri telah beralih ke multicore processors dengan beberapa prosesor sederhana pada satu chip untuk memanfaatkan peningkatan kepadatan transistor [8].

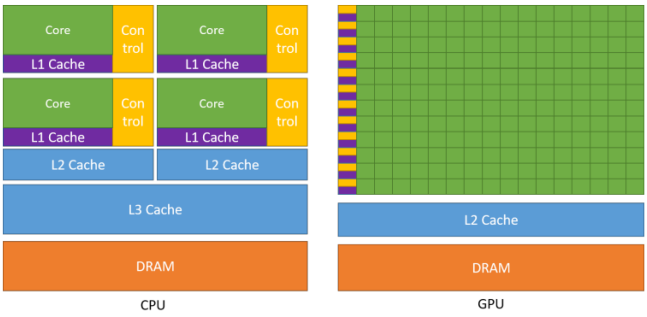
Pemrograman paralel adalah metode pengembangan perangkat lunak yang menggunakan sumber daya komputasi dengan membagi eksekusi program menjadi unit-unit yang dapat berjalan bersamaan. Tujuannya adalah meningkatkan kinerja komputasi dan mempercepat penyelesaian tugas-tugas komputasi berat. Teknik-teknik seperti multithreading, pemrosesan terdistribusi, dan penggunaan multiple core atau prosesor memungkinkan komputer menjalankan beberapa instruksi secara bersamaan, menghasilkan percepatan signifikan dalam pemrosesan data.

Terdapat dua pendekatan yang umum digunakan dalam paralelisme: paralelisme tugas dan paralelisme data. Dalam paralelisme tugas, kita membagi berbagai tugas yang dilakukan dalam menyelesaikan masalah di antara core-core tersebut. Dalam paralelisme data, kita membagi data yang digunakan dalam menyelesaikan masalah di antara core-core tersebut, dan setiap core melaksanakan operasi yang lebih atau kurang serupa pada bagian datanya [8].

1) *Unit Pemrosesan Grafis (GPU)*

GPU, atau Graphics Processing Unit, adalah sebuah unit pemrosesan khusus yang dirancang untuk mengelola dan memproses data yang berkaitan dengan grafis dan visualisasi. Pengolahan satu gambar dapat memerlukan jumlah data yang sangat besar sehingga unit pemrosesan grafis (GPU) perlu menjaga laju pergerakan data yang sangat tinggi, dan untuk menghindari hambatan pada akses memori, mereka sangat bergantung pada multithreading perangkat keras; beberapa sistem mampu menyimpan status dari lebih dari seratus thread yang ditangguhkan untuk setiap thread yang sedang dieksekusi [8].

Unit Pemrosesan Grafis (GPU) memberikan throughput instruksi dan bandwidth memori yang jauh lebih tinggi dibandingkan dengan CPU dalam kerangka harga dan daya yang serupa [4]. Perbedaan kapabilitias ini ada karena CPU dan GPU dirancang untuk hal yang berbeda. CPU dirancang untuk menjalankan sebuah sekuens operasi secepat mungkin. Sebuah CPU bisa mengeksekusi beberapa core dalam satu waktu (paralel). Sedangkan GPU didesain untuk mengeksekusi ribuan thread secara paralel. Karena hal tersebut, GPU lebih sering digunakan untuk pemrosesan data daripada CPU yang sering digunakan untuk flow control. Gambar II.2 menunjukkan perbandingan antara jumlah transistor pada CPU dan GPU NVIDIA.



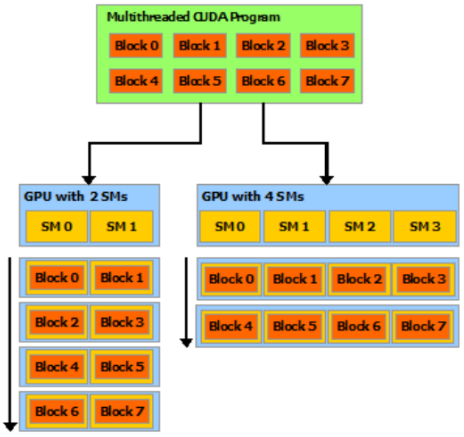
Gambar II.2. Perbandingan Transistor pada CPU dan GPU [4]

2) *CUDA*

CUDA (Compute Unified Device Architecture) adalah platform pemrograman dan kerangka kerja pengembangan yang dikembangkan oleh NVIDIA untuk pemrograman paralel pada GPU (NVIDIA, 2023). CUDA dirancang untuk memaksimalkan potensi pemrosesan paralel yang tinggi yang dimiliki oleh GPU, dengan fokus pada aplikasi pemrosesan data dan kecerdasan buatan.

Tantangan dalam sistem paralel adalah mengembangkan perangkat lunak aplikasi yang secara transparan meningkatkan tingkat paralelismenya untuk memanfaatkan peningkatan jumlah inti pemrosesan (NVIDIA, 2023). CUDA dirancang untuk mengatasi tantangan ini, dengan tiga kunci abstraksi yaitu hierarki thread, shared memories, dan barrier synchronization. Abstraksi-abstraksi tersebut membantu programmer untuk melakukan partisi terhadap suatu masalah menjadi sub-masalah yang bisa diselesaikan secara independen secara paralel dalam thread block. CUDA mendukung automatic scalability, yang artinya thread block dapat dijadwalkan pada salah satu dari seluruh

microprocessor, dalam urutan bebas, dan konkuren maupun sekuensial. Sehingga program CUDA dapat dieksekusi dalam GPU berbeda yang memiliki jumlah multiprocessor yang berbeda. Ilustrasi skalabilitas dari program CUDA dapat dilihat pada Gambar II.3.



Gambar II.3. Ilustrasi Skalabilitas dari Program CUDA [4]

III. RANCANGAN SOLUSI

Cheon dkk. (2018) memperkenalkan sebuah pendekatan inovatif pada skema enkripsi homomorfik aproksimatif Cheon-Kim-Kim-Song (CKKS) dengan mengintegrasikan sistem Residue Number System (RNS). Makalah yang berjudul 'A Full RNS Variant of Approximate Homomorphic Encryption' membahas secara mendalam mengenai adaptasi RNS dalam mempercepat proses komputasi CKKS. Cheon dkk. (2019) berkontribusi pada menyediakan kode sumber yang mendukung implementasi praktis dari varian RNS CKKS yang mereka kembangkan. Ketersediaan kode sumber ini membuka peluang bagi para peneliti untuk melakukan eksperimen dan validasi dari teori yang diajukan.

Selain itu, terdapat juga sebuah pustaka publik CDKS yang tersedia secara online. Kedua sumber ini menjadi landasan untuk eksplorasi lebih lanjut dalam meningkatkan efisiensi dan performa skema enkripsi homomorfik, khususnya dalam implementasi pada perangkat GPU. Kedua pustaka enkripsi homomorfik ini akan digunakan sebagai pembanding dalam implementasi serial Multi-Key CKKS.

Desain implementasi dari skema Chen-Dai-Kim-Song (CDKS) untuk penggunaan serial menggunakan pustaka Standard Template Library (STL) dari bahasa pemrograman C++. Struktur data yang digunakan meliputi vektor, map, dan complex, semuanya merupakan komponen dari STL. Terdapat perbedaan signifikan antara skema CKKS tradisional dan varian berbasis Residue Number System (RNS) dalam penggunaan struktur data.

Dalam skema CKKS konvensional, beberapa tingkat atau level modulus ditentukan, dinotasikan sebagai Q0, Q1, ..., Qn. Setiap operasi aritmetika hanya melibatkan satu polinomial pada suatu waktu, dengan modulus yang merupakan hasil perkalian semua level modulus dari Q0 hingga Qn. Karena setiap modulus biasanya adalah bilangan 60-bit, akumulasi perkalian modulus ini menghasilkan bilangan modulus yang sangat besar, meningkatkan penggunaan memori. Ukuran total Q bisa mencapai 800 bit.

Untuk mencapai percepatan maksimal dan mengurangi overhead, semua komputasi dan struktur data yang diperlukan disimpan pada GPU. Struktur data ini menggunakan pustaka CUDA Thrust, khususnya kelas 'DeviceVector' yang diimplementasikan oleh Jung dkk. [9].

Berbeda dengan CKKS konvensional, varian CKKS yang menggunakan RNS mengelola n polinomial, masing-masing sesuai dengan jumlah level modulus. Pendekatan ini secara signifikan mengurangi kebutuhan memori karena semua operasi dapat dieksekusi dalam batas integer 64-bit. Dalam skema paralelisasi, desain utama melibatkan penyimpanan array satu dimensi yang mewakili kumpulan polinomial untuk setiap modulus tertentu. Hal ini memungkinkan eksekusi operasi secara paralel pada setiap koefisien secara independen. Untuk operasi perkalian dalam polinomial, implementasi memanfaatkan Number Theoretic Transform (NTT), yang memfasilitasi perkalian polinomial secara point-wise, mempercepat proses tersebut dengan mengurangi kompleksitas perkalian dari kuadratik O(n^2) menjadi linear O(n).

A. *Pembangkitan Kunci*

Tahap pembangkitan kunci merupakan tahap awal dari skema CKKS. Pembangkitan kunci terdiri dari tiga tahap yaitu pembangkitan kunci privat, kunci publik, dan kunci evaluasi. Kunci privat dibangkitkan secara random, dengan letak koefisien random serta koefisien yang random. Pembangkitan kunci privat kurang memiliki kompleksitas yang tinggi sehingga tidak diimplementasikan secara paralel.

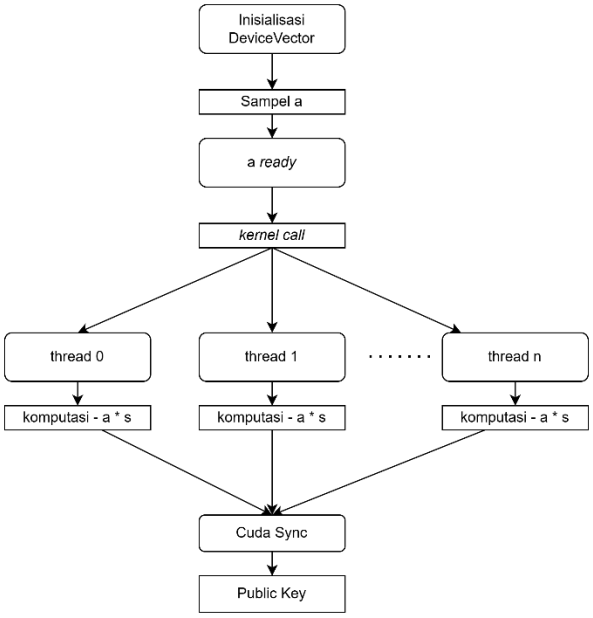
Formula pembangkitan kunci publik adalah pk ←(b,a) yang dalam hal ini b ← -a.s+e. Variabel a merupakan polinomial yang dibangkitkan secara acak. Paralelisasi pada pembangkitan kunci publik ini adalah pada operasi negasi dan perkalian tersebut.

Algorithm 1 CUDA Kernel for Modular Negative Multiplication

```
1: procedure modNegMulKernel(a, b, result, mod, N, M)
2:     i ← threadIdx.x + blockIdx.x × blockDim.x
3:     if i < N × M then
4:         r ← mod[i/N] − a[i]
5:         result[i] ← operation::modMultiply(r, b[i], mod[i/N])
6:     end if
7: end procedure
```

Gambar III.1 Kernel Pembangkitan Kunci Publik

Gambar III.1 menunjukkan fungsi kernel yang berjalan pada GPU untuk proses pembangkitan kunci publik. Setiap thread akan mengatasi satu buah koefisien polinomial dan melakukan operasi negasi (implementasi menggunakan unsigned integer, sehingga operasi negasi dilakukan dengan Modulus – X) serta perkalian dengan kunci privat. Perkalian ini juga menggunakan operasi modular untuk setiap level. Ilustrasi umum operasi penjumlahan secara paralel dapat dilihat pada gambar III.2.



Gambar III.2 Diagram Operasi Pembangkitan Kunci Publik Paralel

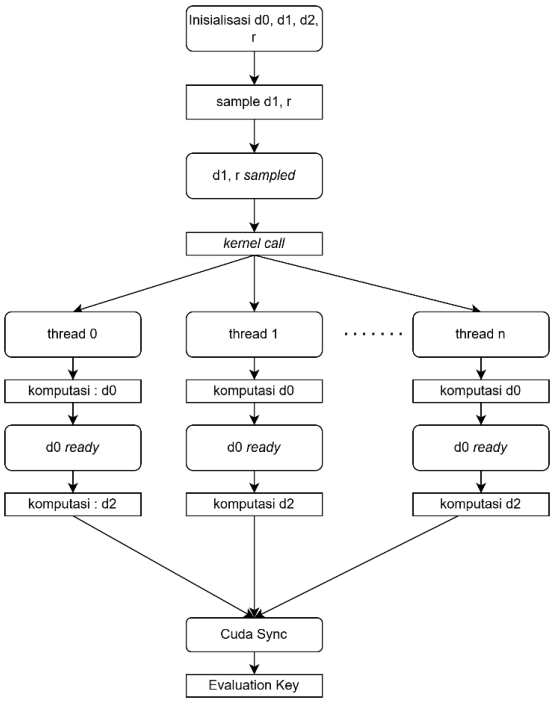
Pembangkitan kunci evaluasi memerlukan langkah yang lebih panjang daripada kunci-kunci sebelumnya. Pembangkitan kunci evaluasi skema CKKS melibatkan kunci privat dan kunci publik yang telah dibuat. Hasil akhir kunci publik memiliki tiga buah variabel yaitu d0, d1, dan d2.

Gambar III.3 menunjukkan rancangan paralel dari pembangkitan kunci evaluasi dari skema CKKS. Seperti pada pembangkitan kunci publik sebelumnya, setiap thread mengatasi tiap koefisien dari array-array yang dimasukkan. Gambaran umum proses pembangkitan kunci evaluasi secara paralel terdapat pada Gambar III.4.

Algorithm 2 CUDA Kernel for Evaluation Key Computation

```
1: procedure EVALUATIONKEYKERNEL(d0, d1, d2, r, s, pModQ, d, pub, N, q, p)
2:     i ← threadIdx.x + blockIdx.x × blockDim.x
3:     if i < N × (p + q) then
4:         t ← d[i/N] − d1[i]
5:         d0[i] ← modMultiply(t, s[i], d[i/N])
6:         d2[i] ← modMultiply(r[i], pub[i], d[i/N])
7:         if i ≥ N × p then                          ▷ q exclusive
8:             t ← modMultiply(r[i], pModQ[(i − (N × p))/N], d[i/N])
9:             d0[i] ← modAdd(d0[i], t, d[i/N])
10:            t ← modMultiply(s[i], pModQ[(i − (N × p))/N], d[i/N])
11:            d2[i] ← modAdd(d2[i], t, d[i/N])
12:        end if
13:    end if
14: end procedure
```

Gambar III.3 Kernel CUDA Pembangkitan Kunci Evaluasi



Gambar III.4 Diagram Operasi Pembangkitan Kunci Evaluasi Paralel

B. Enkripsi dan Dekripsi

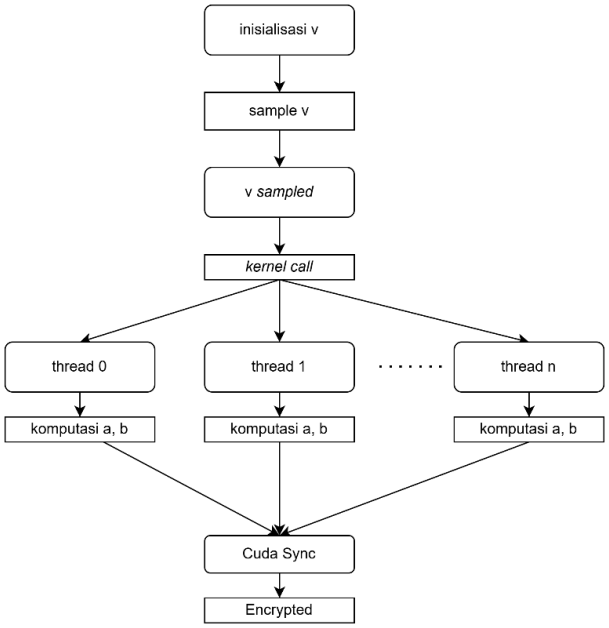
Proses enkripsi dan dekripsi hanya melibatkan modulus Q saja. Pada dasarnya, proses enkripsi melakukan pembangkitan sebuah polinomial v. Proses pembangkitan ini sedikit lebih unik daripada proses random sebelumnya. Proses pembangkitan v menelusuri setiap koefisien polinomial sampai pada derajat terakhir, pada setiap koefisien, dibangkitkan sebuah angka dari set {-1, 0, 1} secara uniform random. Kemudian dari v tersebut dikalikan dengan kunci publik yang akhirnya ditambahkan dengan plaintext. Operasi perkalian dan penjumlahan inilah yang diparalelkan dalam GPU.

Algorithm 3 CUDA Kernel for Element-wise Encryption

```
1: procedure ENCRYPTKERNEL(v, p, a, b, N, sz)
2:     i ← threadIdx.x + blockIdx.x × blockDim.x
3:     level ← i/N                    ▷ Determine the level for modulus selection
4:     if i < sz then
5:         r ← modMultiply(a[i], v[i], constQ[level])
6:         a[i] ← modAdd(r, p[i], constQ[level])
7:         b[i] ← modMultiply(b[i], v[i], constQ[level])
8:     end if
9: end procedure
```

Gambar III.5 Kernel CUDA Enkripsi Plaintext

Kernel CUDA untuk proses enkripsi terdapat pada Gambar III.5. Setiap thread pada GPU akan memproses koefisien-koefisien dari tiga buah array ini. Level dari indeks ditentukan dengan i / N, karena setiap array v, p, dan a merupakan sebuah array satu dimensi yang secara implisit mewakili array dua dimensi. Jumlah elemen pada setiap array adalah N * level, dengan N adalah derajat polinomial. Gambaran umum proses enkripsi paralel terdapat pada Gambar III.6.



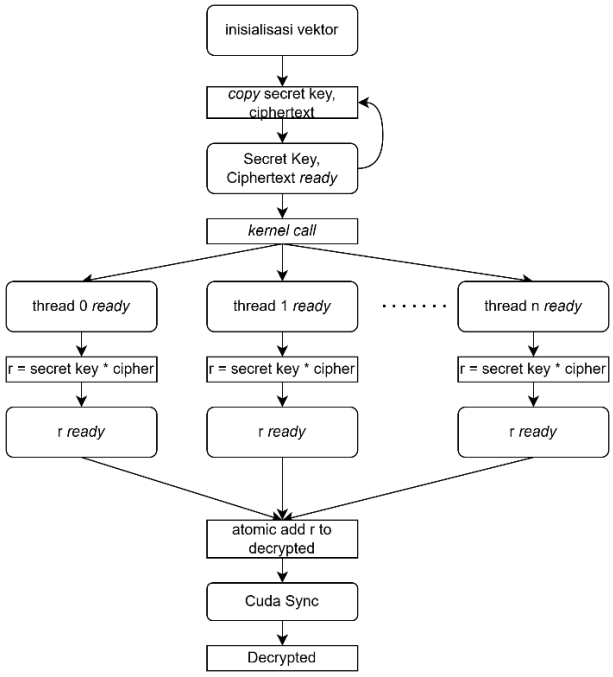
Gambar III.6 Diagram Ilustrasi Enkripsi Paralel

Proses dekripsi yang diimplementasikan adalah proses dekripsi secara menyeluruh, dengan masukan ciphertext, dan semua secret key yang pernah terlibat dalam ciphertext. Pada intinya, proses dekripsi adalah menjumlahkan semua hasil kali ciphertext dengan kunci privat yang terlibat menjadi satu buah polinomial. Untuk proses perkalian antara ciphertext dan kunci privat dapat dilakukan paralelisai sederhana seperti algoritma-algoritma sebelumnya dengan mengalikan koefisien yang berkorespondensi. Operasi penjumlahan ini sedikit lebih memerlukan penanganan khusus karena operasi reduksi yang menjumlahkan seluruh polinomial menjadi satu. Kernel proses dekripsi terdapat pada gambar III.7, sedangkan gambaran umum proses dekripsi paralel terdapat pada gambar III.8.

Algorithm 4 CUDA Kernel for Parallel Decryption

```
1: procedure DECRYPTKERNEL(decrypted, givenPolys, secretPolys, q, N, numIds)
2:     i ← threadIdx.x + blockIdx.x × blockDim.x      ▷ Calculate global index
3:     if i < N × (numIds + 1) then
4:         index ← i%N      ▷ Determine the position within the polynomial
5:         r ← modMultiply(givenPolys[i], secretPolys[i], q)      ▷ Modular multiplication
6:         atomicAdd64(&decrypted[index], r%q)      ▷ Atomic modular addition
7:     end if
8: end procedure
```

Gambar III.7 Kernel CUDA Proses Dekripsi (refer)



Gambar III.8 Diagram Ilustrasi Dekripsi Paralel

C. *Operasi Homomorfik*

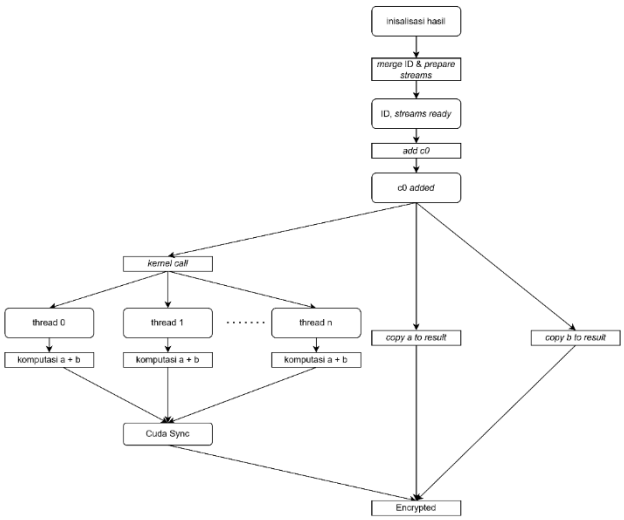
Operasi homomorfik penjumlahan pada skema CDKS pada dasarnya adalah menggabungkan dua buah ciphertext. Jika ditemukan sebuah cipher yang memiliki ID yang sama dengan cipher yang lain, kedua buah cipher tersebut dijumlahkan. Jadi paralelisasi penjumlahan secara sederhana dapat dilakukan tiap koefisien polinomial per thread seperti pada Gambar III.9.

Algorithm 5 CUDA Kernel for Element-wise Modular Addition

```
1: procedure ADDKERNEL(a, b, q, N, l)
2:     i ← threadIdx.x + blockIdx.x × blockDim.x
3:     if i < N × l then
4:         a[i] ← (a[i] + b[i])%q[i/N]
5:     end if
6: end procedure
```

Gambar III.9 Kernel CUDA Penjumlahan Modular

Kernel ini hanya akan dipanggil ketika ditemukan sebuah ID kunci yang hadir pada kedua ciphertext. Jika kunci hanya ada pada salah satu, dilakukan copy dari ciphertext ke hasil. Gambar III.10 menunjukkan ilustrasi umum proses penjumlahan secara paralel.



Gambar III.10 Diagram Ilustrasi Operasi Penjumlahan

Operasi perkalian adalah operasi yang paling rumit dari seluruh operasi-operasi sebelumnya juga memakan waktu paling lama. Proses awal dari perkalian adalah menyamakan dasar ID pada kedua buah ciphertext. Dalam hal ini, kumpulan ID dalam kedua ciphertext akan di-merge dan berdasarkan ID tersebut kedua ciphertext akan diurutkan. Fungsi akan mengembalikan dua buah ciphertext yang akan digunakan pada langkah berikutnya. Terdapat kernel-kernel kecil yang digunakan dalam proses perkalian ini. Kernel-kernel tersebut digunakan untuk operasi polinomial agar bisa perkalian atau penjumlahan bisa dilakukan secara konstan.

Operasi perkalian akan menggunakan ketiga kernel pada Gambar III.11, Gambar III.12, dan Gambar III.13. Fungsi-fungsi kernel ini dibuat secara spesifik untuk meningkatkan performa dan mengurangi overhead.

Algorithm 7 CUDA Kernel for Modular Multiplication

```
procedure MODMULKERNEL(a, b, result, mod, N, M)
    i ← threadIdx.x + blockIdx.x × blockDim.x
    if i < N × M then
        result[i] ← modMultiply(a[i], b[i], mod[i/N])
    end if
end procedure
```

Gambar III.11 Kernel CUDA Perkalian Modular

Algorithm 8 CUDA Kernel for Modular Multiplication and Addition

```
procedure MODMULANDADDKERNEL(a, b, result, mod, N, M)
    i ← threadIdx.x + blockIdx.x × blockDim.x
    if i < N × M then
        r ← modMultiply(a[i], b[i], mod[i/N])
        result[i] ← modAdd(r, result[i], mod[i/N])
    end if
end procedure
```

Gambar III.12 Kernel CUDA Perkalian dan Penjumlahan

Algorithm 9 CUDA Kernel for Modular Multiplication and Addition

```
procedure MODMULEVALKEYS(a, b, c, d0, d1, d2, c0, ci, cj, mod, N, M)
    i ← threadIdx.x + blockIdx.x × blockDim.x
    if i < N × M then
        a[i] ← modMultiply(a[i], d0[i], mod[i/N])
        b[i] ← modMultiply(b[i], d1[i], mod[i/N])
        c[i] ← modMultiply(c[i], d2[i], mod[i/N])
        c0[i] ← modAdd(c0[i], a[i], mod[i/N])
        ci[i] ← modAdd(ci[i], b[i], mod[i/N])
        cj[i] ← modAdd(cj[i], c[i], mod[i/N])
    end if
end procedure
```

Gambar III.13 Kernel CUDA Untuk Kunci Evaluasi

IV. PENGUJIAN

A. *Lingkungan Pengujian*

Pengembangan perangkat lunak dilakukan pada lingkungan dengan spesifikasi sebagai berikut.

1) Perangkat Keras :

- Processor : AMD Ryzen 9 5950X 16-Core Processor
- Memory : 32 GB DDR4
- Spesifikasi GPU :
  - Nama : NVIDIA GeForce RTX 3090 Ti
  - Jumlah Streaming Multiprocessor : 84
  - Jumlah CUDA Cores : 10752
  - Ukuran memori : 23.58 GiB

2) Perangkat Lunak

- Sistem Operasi : Ubuntu 22.04
- Compiler : NVCC 12.5, G++ 11.3
- IDE : Visual Studio Code

B. *Tujuan Pengujian*

Pengujian kinerja implementasi algoritma CKKS dilaksanakan untuk mengukur waktu eksekusi dengan variabel tinggi derajat cincin polinomial. Implementasi serial dan paralel diukur dalam waktu eksekusi dan dianalisis untuk menentukan percepatan yang dicapai. Tujuan utama pengujian ini adalah untuk menentukan apakah paralelisasi algoritma CKKS menggunakan CUDA berhasil mempercepat eksekusi operasi-operasi homomorfik.

C. *Hasil Pengujian*

1) Pengujian 2 Kunci

Pengujian pertama dilakukan dengan dua kunci yang berbeda untuk setiap operasi. Pengukuran waktu eksekusi dilakukan pada saat sebelum dan sesudah operasi dengan presisi 1 ms. Tabel IV.1 menunjukkan hasil dari pengujian implementasi paralel dua kunci berbeda.

Tabel IV.1 Hasil Pengujian Dua Kunci

Operasi	$\log_2(N)$	Waktu Eksekusi (ms)		*Speed Up*
		Serial	Paralel	
Pembangkitan Kunci Publik	14	640.2552	13.13454	48.7459
	15	1401.75	27.31802	51.3123

	16	2934.752	57.21806	51.2907
Pembangkitan Kunci Evaluasi	14	1210.22	28.3747	42.6514
	15	2657.936	59.13276	44.9486
	16	5647.724	123.6662	45.6691
Enkripsi	14	1204.746	14.23854	84.6116
	15	2632.766	28.47434	92.4610
	16	5674.936	58.11496	97.6502
Dekripsi	14	31.97194	1.337528	23.9038
	15	71.31378	2.05646	34.6779
	16	148.58	3.842302	38.6695
Penjumlahan	14	3.929588	0.683422	5.7499
	15	10.025966	0.8556668	11.7171
	16	24.95932	1.147714	21.7470
Perkalian	14	10069.326	135.884	74.1024
	15	21597.16	283.1816	76.2661
	16	45145.46	581.9354	77.5781

2) Pengujian Banyak Kunci

Semakin banyak *party* yang terlibat dalam komputasi, semakin banyak pula jumlah kunci yang diperlukan. Sehingga, beban komputasi untuk banyak kunci juga akan meningkat. Pengujian selanjutnya dilakukan dengan mengukur waktu eksekusi jika jumlah user meingkat. Tabel IV.2 menunjukkan hasil pengujian dengan banyak user.

Tabel IV.2 Hasil Pengujian Banyak Kunci

Operasi	Jumlah *Party/User*	Waktu Eksekusi (ms)		*Speedup*
		Serial	Paralel	
Penjumlahan	8	39.1881	2.31279	16.9441
	16	80.5584	4.12945	19.5083
	32	147.239	7.99695	18.4119
Perkalian	8	328649	3803.96	86.3965
	16	1259050	14627.8	86.0724
	32	4956800	57694.8	85.9142
Dekripsi	8	33.9043	2.40029	14.1251
	16	38.3332	2.88483	13.2879
	32	41.6133	3.82528	10.8785

3) Occupancy GPU

Occupancy GPU menunjukkan rasio antara antara jumlah warp aktif per multiprocessor dengan jumlah maksimum warp yang bisa aktif. Occupancy tinggi dapat menyembunyikan latensi dan meningkatkan paralelisme, walaupun tidak selalu menjamin performa optimal. Analisis occupancy dilakukan per kernel untuk mengetahui efisiensi penggunaan core CUDA. Tabel IV.3 menunjukkan *occupancy* GPU tiap kernel.

Tabel IV.3 Occupancy GPU Per Kernel

Kernel	Occupancy (%)
bitReverseBatchKernel	14.31
toNTTBatchParallelKernel	59.9
NTTPar	66.68
bitReverseKernel	15.05
fromNTTBatchParallelKernel	59.25
iNTTPar	66.59
samplePolynomialKernel	16.09
modNegMulKernel	71.01
evaluationKeyKernel	69.75
encryptKernel	65.75
decryptKernel	24.1
modAddKernel	61.05
modMulKernel	66.81
modUpParallel	65.21
modDownParallel	65.26
modMulEvalKeys	72.36
modMulAndAddKernel	65.51

rescaleKernel	65.36

V. KESIMPULAN

Implementasi enkripsi homomorfik total skema Chen-Dai-Kim-Song (CDKS) pada GPU dengan menggunakan CUDA menunjukkan peningkatan kinerja yang signifikan. Penerapan komputasi pada GPU serta penurunan kompleksitas operasi-operasi polinomial berhasil dioptimalkan, menghasilkan waktu eksekusi yang lebih efisien dibandingkan dengan implementasi serial.

Hasil pengujian menunjukkan bahwa pembangkitan kunci publik secara paralel mencapai percepatan sebesar 50 kali dibandingkan dengan metode serial. Selain itu, pembangkitan kunci evaluasi paralel menunjukkan percepatan hingga 40 kali. Proses enkripsi paralel juga mencatat percepatan yang sangat signifikan, yaitu hingga 90 kali.

Proses dekripsi paralel mencapai percepatan hingga 30 kali, sementara proses penjumlahan paralel mencapai percepatan hingga 15 kali. Proses perkalian paralel menunjukkan percepatan hingga 70 kali. Operasi-operasi dengan banyak kunci secara paralel juga mencatat percepatan yang signifikan, mencapai hingga 85 kali. *Occupancy* GPU yang tinggi menunjukkan kernel-kernel CUDA menggunakan CUDA cores secara efektif. Peningkatan kinerja yang dicapai menunjukkan bahwa penggunaan GPU dan metode paralelisasi dalam skema CDKS efektif.

UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Allah SWT, atas rahmat-Nya sehingga penulisan makalah ini dapat diselesaikan. Penulis menyampaikan rasa terima kasih yang mendalam kepada orang tua, atas dukungan dan doa yang tiada henti. Terima kasih juga disampaikan kepada Dr. Ir. Rinaldi Munir, M.T. dan Dr. Eng. Infall Syafalni, S.T., M.Sc. selaku pembimbing, yang telah memberikan bimbingan dalam pengerjaan makalah. Ucapan terima kasih juga disampaikan kepada seluruh teman-teman yang telah menemani dan membantu penulis selama masa perkuliahan.

REFERENSI

[1] Yi, X., Paulet, R., & Bertino, E. (2014). Homomorphic Encryption and Applications. New York, NY, USA: Springer, 2014.

[2] Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2016). Homomorphic encryption for arithmetic of approximate numbers. IACR Cryptology ePrint Archive, 2016, 421.

[3] Chen, G., Dai, W., Kim, M., & Song, Y. (2019). Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference. IACR Cryptology ePrint Archive, 2019, 524.

[4] NVIDIA. (2023). CUDA Toolkit Documentation. Diakses dari https://docs.nvidia.com/cuda/index.html

[5] Reynaldi, D. (2023). Perancangan dan Percepatan Algoritma Enkripsi Homomorfik Total Skema Cheon, Kim, Kim, Song (CKKS) dengan GPU. Institut Teknologi Bandung.

[6] Özcan, A. Ş., Ayduman, C., Türkoğlu, E. R., & Savaş, E. (2022). Homomorphic Encryption on GPU. Cryptology ePrint Archive, 2022, 1222.

[7] Hennessy, J. L., & Patterson, D. A. (2017). Computer architecture: a quantitative approach. Morgan Kaufmann.

[8] Pacheco, P. (2011). An introduction to parallel programming. Morgan Kaufmann.

[9] Jung, W., Kim, S., Ahn, J. H., Cheon, J. H., & Lee, Y. (2021). Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs. IACR Cryptology ePrint Archive, 2021, 508.