Akselerasi SSL pada OpenSSL Menggunakan GPU Berbasiskan OpenSSL EVP API

Adityo Jiwandono and 13507015[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*jiwandono@arc.itb.ac.id*

***Abstrak*—Kecepatan transfer data pada semakin meningkat seiring dengan berkembangnya teknologi komputer. Hal ini berlaku pula pada jaringan komputer. Sebagian besar laptop dan komputer desktop telah memiliki *port* Ethernet yang menyediakan *bandwidth* hingga 1 Gbps. Standar Ethernet terbaru bahkan sudah mendukung *bandwidth* sebesar 100 Gbps. Walaupun demikian, pengguna jaringan tidak hanya membutuhkan kecepatan namun juga keamanan. Faktor keamanan ini dipenuhi dengan mengimplementasikan algoritma kriptografi. Salah satu standar algoritma kriptografi saat ini adalah *Advanced Encryption Standard* (AES). Akan tetapi, penambahan aspek keamanan ini menimbulkan *overhead* pada pemrosesan data sehingga *throughput* jaringan tidak optimal. Makalah ini mengusulkan metode untuk mengurangi *overhead* tersebut dan meningkatkan performa kriptografi AES dengan memanfaatkan kemampuan komputasi paralel GPU yang diimplementasikan ke dalam OpenSSL berbasiskan EVP API. *Throughput* yang diperoleh pada makalah ini adalah 6,24 Gbps menggunakan GPU NVIDIA Geforce GTX 460. Percepatan yang dihasilkan adalah sebesar 7,8 kali dibandingkan pada implementasi OpenSSL 1.0.0d.**

***Kata kunci*—AES, CUDA, GPGPU, SSL.**

I. Latar Belakang

Kemajuan teknologi komputer telah mengakibatkan perkembangan pada banyak hal. Salah satunya adalah penigkatan kecepatan transfer data pada jaringan komputer. Standar teknologi jaringan komputer Ethernet telah meningkatkan *bandwidth* mulai dari Ethernet 1 Mbps dan 10 Mbps, Fast Ethernet (100 Mbps), Gigabit Ethernet (1 Gbps), dan standar terbaru saat ini yang mencapai 100 Gbps.

Kecepatan transfer data pada jaringan komputer telah meningkat dengan pesat. Apalagi Sebagian besar komputer desktop dan laptop saat ini sudah dilengkapi dengan *port* Gigabit Ethernet. Walaupun begitu, pengguna jaringan tidak hanya membutuhkan faktor kecepatan saja. Pengguna juga membutuhkan keamanan data. Faktor keamanan ini diwujudkan dalam bentuk protokol dan algoritma kriptografi. Protokol kriptografi yang saat ini dignuakan adalah *Secure Sockets Layer* (SSL) dan *Transport Layer Security* (TLS). *Advanced Encryption Standard* (AES) adalah salah satu standar algoritma kriptografi simetri dan RSA adalah algoritma kriptografi kunci publik yang saat ini banyak digunakan. OpenSSL adalah implementasi SSL dan TLS yang bersifat *open source*. Dengan demikian, faktor keamanan telah terpenuhi.

Penerapan algoritma kriptografi pada jaringan memang telah berhasil memenuhi kebutuhan akan keamanan data. Permasalahan yang muncul adalah adanya *overhead* akibat pemrosesan pada algoritma kriptografi yang mengakibatkan tidak optimalnya *throughput* jaringan komputer. Padahal, kebutuhan solusi kriptografi yang efisien terus meningkat. Untuk mengatasi permasalahan tersebut, muncul ide untuk mengakselerasi performa algoritma kriptografi pada SSL dengan memanfaatkan kemampuan komputasi paralel pada GPU.

II. Penelitian Terkait

Solusi akselerasi SSL dalam bentuk hardware telah lama dilakukan. Salah satu bentuk solusi tersebut adalah SSL *card* yang dapat dipasang pada slot PCI pada *mainboard*. Solusi ini telah ditinggalkan karena performa SSL-nya semakin tertinggal oleh solusi SSL yang diimplementasikan pada CPU. Solusi hardware lainnya adalah dengan menggunakan ASIC dan FPGA yang dapat memberikan *throughput* SSL AES hingga 70 Gbps yang lakukan oleh Hodjat dkk. [1] dengan mengoptimasi S-box dan kombinasi antara S-box dan operasi MixColumn.

Solusi berbasis perangkat keras lainnya adalah adanya instruksi khusus pada prosesor VIA C3. Prosesor ini mamuat instruksi khusus yang mengangani algoritma AES sehingga dapat meningkatkan performa AES. *Throughput* yang dicapai menggunakan prosesor ini adalah 6,2 Gbps [9].

GPU sendiri telah menjadi subjek penelitian untuk komputasi paralel di berbagai bidang. Shuai Che dkk. [10] telah melakukan studi performa sejumlah pemanfaatan GPU untuk permasalahan yang sifatnya umum, non-grafis, menggunakan arsitektur NVIDIA CUDA. Pada makalah tersebut dipaparkan bahwa penggunaan GPU untuk menyelesaikan permasalahan yang dipilih telah memberikan peningkatan performa yang signifikan. Owens [4] juga telah melakukan survei serupa dan menyimpulkan bahwa komputasi GPU memberikan peluang besar untuk menjadikannya sebagai co-prosesor

komputasi paralel.

Gu Liu [5] telah melakukan studi terhadap sejumlah algoritma kriptografi cipher blok, salah satunya adalah AES. Pada studi tersebut implementasi algoritma AES pada GPU menghasilkan *throughput* sebesar sekitar 4,2 Gbps. GPU yang digunakan adalah NVIDIA Geforce 9800 GTX. Manavski [8] berhasil memperoleh *throughput* sebesar 8,28 Gbps menggunakan GPU NVIDIA Geforce 8800 GTX.

Solusi hardware menggunakan ASIC dan FPGA memberikan *throughput* yang paling besar namun tidak lebih mudah dilakukan daripada menggunakan GPU. Saat ini GPU telah menjadi barang komoditas dan dapat diperoleh dengan mudah.

Walaupun penelitian tentang kriptografi pada GPU telah banyak dilakukan dan diimplementasikan, integrasi belum dilakukan pada aplikasi OpenSSL yang merupakan salah satu implementasi SSL dan TLS *open source* yang cukup banyak digunakan.

III. Pemrograman GPU

A. Pemrograman GPU tradisional dan GPGPU

Secara umum, GPU memiliki tiga tahap pemrosesan dan pada tiap tahapnya ditangani oleh prosesor tertentu yaitu *rasterizer*, *vertex processor*, dan *fragment processor*. *Rasterizer* memiliki fungsi yang tetap sementara *vertex processor* dan *fragment processor*, pada arsitektur yang baru, dapat diprogram. *Vertex processor* memungkinkan sebuah program dapat dijalankan untuk setiap vertex pada objek. Tiap tiga vertex membentuk sebuah segitiga dan dari segitiga inilah fragmen-fragmen dihasilkan. *Fragment processor* melakukan transformasi terhadap fragmen dan kemudian *rasterizer* memprosesnya sehingga siap ditampilkan pada layar. *Texture memory* adalah tempat di mana seluruh data input disimpan. Untuk aplikasi GPGPU, persoalan dan data harus dipetakan ke dalam domain grafis terlebih dahulu. Proses pemetaan persoalan ini tidak selalu mudah untuk dilakukan dan performa akhir sangat dipengaruhi oleh pemetaan yang dipilih.

B. GPGPU menggunakan NVIDIA CUDA

Dua produsen GPU utama, AMD dan NVIDIA, telah memperkenalkan arsitektur untuk mengembangkan komputasi pada GPU, yang masing-masing bernama CUDA dan CTM. Platform ini memiliki pendekatan baru yang dirancang untuk dapat mengakses perangkat GPU secara *native*. Makalah ini hanya akan membahas arsitektur NVIDIA CUDA.

CUDA merupakan perluasan dari subset bahasa C. Salah satu perluasan tersebut adalah adanya jenis fungsi baru yang dinamakan fungsi kernel. Fungsi ini, jika dipanggil, akan dijalankan oleh sejumlah thread pada GPU secara konkuren. Pemrograman GPU melalui CUDA dapat dipandang sebagai sebuah perangkat yang dapat menjalankan sejumlah besar thread secara paralel.

Satuan eksekusi terkecil pada CUDA adalah thread. Satu thread menangani sebuah kernel. Setiap thread memiliki akses terhadap memori lokal dalam thread itu sendiri. Sejumlah thread dikelompokkan dalam sebuah blok. Thread-thread dalam satu blok memiliki *shared memory* yang dapat diakses oleh seluruh thread dalam blok tersebut. Sejumlah blok dikelompokkan dalam satu grid. Setiap blok ditangani oleh sebuah multiprosesor pada GPU dan satu grid ditangani oleh satu GPU.

Selain memori lokal dan *shared memory* yang telah disebutkan sebelumnya, terdapat juga *register memory*, *global memory*, *constant memory*, dan *texture memory*. Masing-masing jenis memori memiliki karakteristik yang berbeda. Jenis-jenis memori dijelaskan secara lebih rinci pada [7].

IV. Secure Sockets Layer

Secure Sockets Layer (SSL), yang telah diperbarui menjadi *Transport Layer Security* (TLS), adalah protokol kriptografi yang menyediakan keamanan komunikasi yang melalui Internet. SSL dan TLS mengenkripsi sebagian segmen pada koneksi jaringan yang berada di atas layer transport. Enkripsi dilakukan dengan kriptografi simetri untuk kebutuhan privasi dan *message authentication code* serta kuncinya untuk ketepatan pesan.

SSL dan TLS telah digunakan pada berbagai aplikasi. Pada umumnya, SSL dan TLS diimplementasikan di atas protokol layer transport, mengenkapsulasi protokol aplikasi seperti HTTP, FTP, dan SMTP. Salah satu aplikasinya yang populer adalah pada HTTPS yang memberikan jaminan keamanan pada lalu lintas data website. SSL dan TLS dapat juga digunakan untuk membentuk *tunnel* untuk membuat sebuah *Virtual Private Network* (VPN). Salah satu implementasi VPN yang memanfaatkan SSL dan TLS adalah OpenVPN.

V. OpenSSL dan EVP API

OpenSSL adalah salah satu implementasi protokol SSL dan TLS yang *open source*. Di dalamnya sudah terdapat dukungan untuk berbagai fungsi kriptografi, mulai dari kriptografi kunci publik, kriptografi simetri, *message digest*, dan *digital signature* [9].

OpenSSL banyak digunakan oleh aplikasi-aplikasi yang membutuhkan aspek keamanan. Sebagian daftar tersebut dapat dilihat pada [9]. Salah satu contohnya adalah OpenVPN. OpenVPN adalah aplikasi *open source* yang mampu menyediakan layanan VPN. OpenVPN menggunakan OpenSSL untuk keperluan enkripsi dan otentikasi.

OpenSSL didesain dengan sangat moduler. Moduler ini memiliki arti bahwa siapapun dapat menambahkan modul algoritma kriptografi baru pada OpenSSL. OpenSSL EVP (envelope) API menyeragamkan akses terhadap modul-modul kriptografi yang ada.

Jika sebuah modul dipasangkan dengan OpenSSL tanga menggunakan EVP API, algoritma pada modul tersebut

dapat dipakai dengan memanggil fungsi-fungsi yang didefinisikan pada modul tersebut. Penamaan fungsi dan cara pemanggilan sebuah algoritma kriptografi mungkin tidak seragam antara satu modul dengan modul yang lain. Namun jika modul kriptografi diimplementasikan menggunakan EVP API, fungsi-fungsi pada modul tersebut dibungkus oleh API tersebut sehingga cara pengaksesan fungsi dapat diseragamkan dengan hanya memanggil melalui EVP API, tidak perlu memanggil langsung fungsi yang ada pada modul.

`EVP_Seal...` dan `EVP_Open...` menyediakan akses ke fungsi enkripsi dan dekripsi kunci publik untuk mengimplementasikan amplop (envelope) digital. Fungsi `EVP_Sign...` dan `EVP_Verify...` untuk tanda tangan digital. Fungsi `EVP_Encrypt...` dan `EVP_Decrypt...` digunakan untuk mengakses fungsi-fungsi kriptografi simetri. Fungsi `EVP_Digest...` menyediakan fungsi message digest. Fungsi `EVP_PKEY...` menyediakan antarmuka untuk algoritma kriptografi asimetri.

Kelebihan lain penggunaan EVP API pada sebuah modul kriptografi adalah bahwa implementasi sebuah algoritma dapat diganti sehingga sebuah algoritma kriptografi dapat dijalankan oleh *engine* yang berbeda. Misalnya sebuah algoritma AES yang sama dapat diimplementasikan untuk diproses pada CPU atau pada SSL card dengan tetap menjaga penamaan fungsi dan cara aksesnya.

Salah satu lagi kelebihan menggunakan EVP API adalah OpenSSL memiliki mekanisme *benchmark* untuk setiap modul yang diimplementasikan dengan EVP API. Fungsi *benchmark* ini dapat digunakan untuk mengukur performa algoritma kriptografi yang ada pada OpenSSL.

VI. Advanced Encryption Standard

Advanced Encryption Standard (AES) adalah salah satu standar kriptografi sesuai dengan dokumen FIPS 197 dari NIST [6]. AES merupakan algoritma kriptografi simetri yang berbasiskan pada algoritma Rijndael. AES memproses blok data yang berukuran 128 bit. Panjang kunci yang didukung adalah 128, 192, dan 256 bit.

AES memiliki dua proses utama yaitu proses pembangkitan kunci dan proses enkripsi atau dekripsi itu sendiri. Proeses pembangkitan kunci dilakukan pada saat awal sebelum blok-blok mulai diproses untuk dienkripsi. Hasil pembangkitan kunci akan digunakan pada tiap iterasi pada proses enkripsi atau dekripsi. Pada proses enkripsi, plainteks yang berukuran besar dipecah-pecah menjadi banyak blok yang masing-masing berukuran 128 bit. Masing-masing blok plainteks ini akan dimasukkan pada tabel state, berukuran 4×4 byte, yang akan diproses sedemikian rupa sehingga menghasilkan blok cipherteks. Setiap blok plainteks akan diproses dalam sejumlah iterasi round. Jumlah iterasi round ditentukan oleh panjang kunci yang digunakan. Pada setiap iterasi, operasi-operasi yang ada antara lain adalah *AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns*. Detil operasi-operasi tersebut dapat diketahui pada [3].

Untuk prosesor yang mampu memproses data sebesar 32 bit, terdapat optimasi pada operasi-operasi AES sebagaimana dijelaskan pada [3]. Melalui optimasi ini, operasi-operasi AES dapat disederhanakan dengan membagi state menjadi empat dan mengkombinasikan kemungkinan nilai pada setiap iterasi. Nilai-nilai tersebut disimpan pada empat tabel yang masing-masing berukuran 1 KB. Dengan optimasi ini, jumlah operasi per iterasi yang dibutuhkan untuk satu blok plainteks adalah operasi XOR sebanyak 16 kali dan operasi *table lookup* sebanyak 16 kali.

VII. Analisis Paralelisasi SSL-AES

Pada trafik SSL terdapat dua jenis algoritma kriptografi yang terlibat yaitu kriptografi kunci publik dan kriptografi simetri. Kriptografi kunci publik digunakan untuk keperluan otentikasi dan integritas data sedangkan kriptografi simetri digunakan untuk mengamankan data yang ditransmisikan. Berdasarkan keadaan ini, sumber daya komputasi lebih banyak digunakan pada algoritma simetri yang dalam makalah ini adalah AES. Untuk itu, perlu dicari cara yang optimal untuk memanfaatkan GPU sebagai mesin komputasi paralel untuk AES.

Algoritma AES, dengan asumsi menggunakan mode *Electronic Codebook*, secara natural sudah dapat diparalelisasi. Paralelisasi dapat dengan mudah dilakukan karena tidak ada interdependensi antara satu blok dengan blok yang lain. Di dalam tabel state yang dipecah menjadi empat juga tidak ada keterkaitan antara satu bagian state dengan bagian state yang lain. Dengan demikian, paralelisasi dapat dilakukan sampai tingkat pecahan tabel state. Proses iterasi dalam satu blok tidak dapat diparalelisasi karena proses pada suatu iterasi memerlukan hasil pemrosesan dari iterasi sebelumnya. Proses pembangkitan kunci tidak dapat diparalelisasi karena sifat proses ini adalah sangat serial. Selain itu, jumlah data yang diproses sedikit yaitu maksimal sebesar 224 byte.

VIII. Analisis SSL-AES pada GPU

Makalah ini menggunakan GPU NVIDIA Geforce GTX 460. GPU ini memiliki kemampuan CUDA Compute versi 2.1 yang merupakan versi Compute terbaru saat makalah ini ditulis. Device dengan compute 2.1 memiliki karakteristik multiprocessor yaitu sebanyak 48 core CUDA digunakan untuk operasi integer dan floating-point dan terdapat 8 fungsi khusus yang menangani fungsi transenden untuk floating-point presisi tunggal. Kemampuan Compute 2.1 tidak banyak memberikan pengaruh karena pada algoritma AES yang akan diimplementasikan tidak banyak operasi aritmatika namun lebih banyak operasi table lookup.

Referensi implementasi AES yang digunakan pada makalah ini adalah implementasi AES pada OpenSSL [9]. Secara umum, tidak banyak perubahan yang perlu

dilakukan pada algoritma AES untuk CUDA. Pada dasarnya bahasa pemrograman yang ada pada CUDA adalah sama dengan yang digunakan pada implementasi OpenSSL yaitu C. Perubahan yang perlu dilakukan adalah penanganan state pada setiap iterasi round. Pada implementasi CPU, state pada setiap round ditangani secara serial seperti berikut.

```
/* Round 1 */
t0 = Te0[ s0 >> 24        ] ^
     Te1[(s1 >> 16) & 0xff] ^
     Te2[(s2 >>  8) & 0xff] ^
     Te3[ s3        & 0xff] ^
     rk[ 4];
t1 = Te0[ s1 >> 24        ] ^
     Te1[(s2 >> 16) & 0xff] ^
     Te2[(s3 >>  8) & 0xff] ^
     Te3[ s0        & 0xff] ^
     rk[ 5];
t2 = Te0[ s2 >> 24        ] ^
     Te1[(s3 >> 16) & 0xff] ^
     Te2[(s0 >>  8) & 0xff] ^
     Te3[ s1        & 0xff] ^
     rk[ 6];
t3 = Te0[ s3 >> 24        ] ^
     Te1[(s0 >> 16) & 0xff] ^
     Te2[(s1 >>  8) & 0xff] ^
     Te3[ s2        & 0xff] ^
     rk[ 7];
```

dengan `s0`, `s1`, `s2`, `s3` adalah variabel state, `t0`, `t1`, `t2`, dan `t3` adalah variabel temporer untuk pemrosesan state, dan `rk` adalah variabel yang menyimpan hasil pembangkitan kunci.

Pada CUDA, state dapat ditangani secara paralel.Untuk itu, fungsi kernel harus dirancang sehingga satu thread menangani satu elemen state sehingga paralelisasi tercapai. Berikut adalah potongan kode implementasi AES pada GPU untuk satu dari empat elemen state. Setiap elemen state ditangani secara paralel oleh setiap thread.

```
  /* round 1: */
  t[threadIdx.x + 4*threadIdx.y] =
       Te0_gpu[(s[ threadIdx.x          +
4*threadIdx.y] >> 24)       ] ^
       Te1_gpu[(s[(threadIdx.x + 1) % 4 +
4*threadIdx.y] >> 16) & 0xff] ^
       Te2_gpu[(s[(threadIdx.x + 2) % 4 +
4*threadIdx.y] >>  8) & 0xff] ^
       Te3_gpu[(s[(threadIdx.x + 3) % 4 +
4*threadIdx.y]      ) & 0xff] ^
       rd_key[threadIdx.x + 4];
```

Implementasi AES dengan optimasi pada mesin 32-bit pada dasarnya adalah operasi table lookup dan operasi bit. Dengan kata lain, operasi pengaksesan memori dan variabel adalah operasi yang penting untuk dioptimasi. Oleh karena itu, pengelolaan data dan variabel perlu diperhatikan supaya aksesnya optimal. Berikut adalah data dan variabel yang terlibat dalam algoritma AES.

a. Data input.
b. Lookup table.
c. Data hasil pembangkitan kunci key.
d. Variabel state.

Untuk mendapatkan hasil yang optimal, jenis memori yang tepat untuk data dan variabel yang tepat. Berdasarkan karakteristik masing-masing jenis memori, berikut ini adalah analisis terhadap data dan variabel terkait dengan jenis memori yang digunakan. Seluruh data input diletakkan pada global memory. Lookup table karena ukurannya yang tetap dan isinya yang konstan, diletakkan pada constant memory. Untuk meningkatkan performa, lookup table dapat dicoba dipindahkan ke shared memory saat kernel dijalankan. Hasil pembangkitan kunci diletakkan pada global memory dapat dipindahkan ke shared memory atau texture memory untuk meningkatkan performa. Variabel untuk matriks state akan berupa shared variabel pada kernel untuk mempercepat akses.

Selain permasalahan jenis memori yang digunakan, terdapat permasalahan lain yaitu jumlah data input atau buffer yang harus ditransfer dari memori CPU ke memori GPU dan sebaliknya. Ukuran buffer ditentukan oleh pemrogram dan untuk mengetahu ukuran buffer yang optimal harus dilakukan eksperimen untuk masing-masing ukuran buffer yang digunakan.

IX. IMPLEMENTASI PADA GPU

Karena GPU memiliki banyak jenis memori yang dapat digunakan, algoritma akan diimplementasikan menggunakan jenis-jenis memori tersebut untuk mengetahui jenis memori yang tepat. Ukuran buffer yang akan digunakan juga bervariasi mulai 16 KB sampai 8 MB. Berikut adalah tiap-tiap skenario implementasinya. Jumlah thread yang digunakan pada satu blok thread adalah 256.

A. Implementasi hasil adaptasi dari kode OpenSSL

Pada skenario implementasi ini, lookup table disimpan pada constant memory, round key disimpan pada global memory. Round key digunakan sebagai parameter input pada kernel. Variabel temporer untuk menyimpan data input dan hasil pemrosesan tiap round disimpan pada shared memory. Hal ini bertujuan untuk melokalisasi akses terhadap data-data tersebut sehinga diharapkan aksesnya menjadi lebih cepat.

B. Implementasi menggunakan shared memory untuk lookup table

Skenario implementasi ini mirip dengan skenario A. Perbedaanya adalah data pada lookup table yang berada pada constant memory ditransfer ke shared memory. Proses transfer ini berlangsung pada awal kernel dijalankan. Sebelum proses pada setiap round dimulai, thread harus disinkronisasi terlebih dahulu untuk memastikan seluruh proses transfer data dari constant memor ke shared memory selesai. Sinkronisasi dicapai dengan memanggil fungsi `__syncthreads()` tepat setelah operasi transfer.

C. Implementasi menggunakan texture memory untuk round key.

Skenario ini mirip dengan skenario pada A. Perbedaannya adalah data round key diletakkan pada texture memory. Proses transfer data-data tersebut terjadi sebelum kernel dijalankan. Program pada host meminta alokasi texture memory pada device. Setelah memori teralokasi, data-data tersebut dapat diisikan pada texture memory yang telah dialokasikan. Operasi-opreasi pada kernel tidak akan banyak berubah kecuali pada bagian pengaksesan data round key.

D. Implementasi menggunakan shared memory untuk round key.

Skenario ini mirip dengan skenario pada C. Perbedaannya adalah data round key diletakkan pada shared memory.

E. Implementasi dengan menggabungkan implementasi B dan C

Skenario ini adalah gabungan skenario B dan C yaitu meletakkan lookup table pada shared memory dan round key pada texture memory.

F. Implementasi dengan menggabungkan implementasi B dan D

Skenario ini adalah gabungan skenario B dan D yaitu meletakkan semua data-data pendukung pada shared memory.

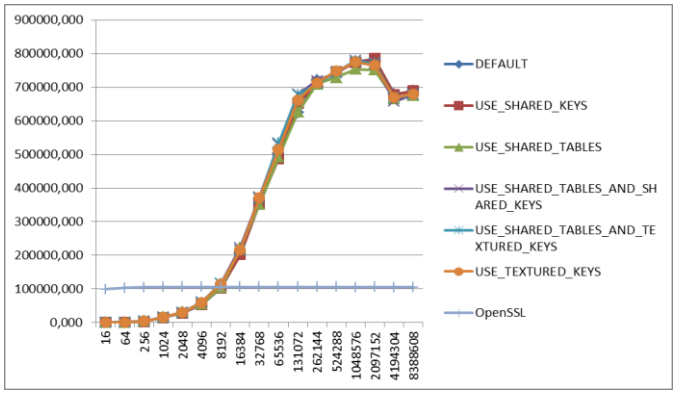
X. PENGUJIAN

Tiap-tiap skenario yang ada pada bagian IX diimplementaskan dan diujikan dengan buffer size yang berbeda-beda. Pengujian dilakukan menggunakan fungsi benchmark pada OpenSSL yang dijalankan dengan perintah `openssl speed -evp aes-128-ecb`. Berikut adalah hasil pengujian tersebut. Sumbu X merupakan ukuran buffer yang digunakan dalam satuan byte, sumbu Y merupakan throughput yang dicapai dalam satuan kilobyte.

Pengujian dilakukan pada komputer dengan spesifikasi sebagai berikut.

- CPU: AMD Phenom II X4 965 3.40 GHz
- Memori: 4 GB DDR3 800 MHz
- GPU: NVIDIA Geforce GTX 460
- OpenSSL: 1.0.0d
- Sistem operasi: Ubuntu 64-bit 10.04

Berdasarkan hasil pengujian, implementasi pada GPU dapat menghasilkan throughput sampai dengan 780 MBps. Ukuran buffer yang optimal berkisar pada 1 MB dan 2 MB. Performa tiap-tiap skenario implementasi pada ukuran buffer tersebut relatif sama, kecuali untuk skenario B yang memberikan performa paling rendah pada ukuran buffer 1 MB dan 2 MB.



Throughput yang diberikan oleh OpenSSL berkisar pada 100 MB/s untuk semua ukuran buffer. Dengan demikian, percepatan yang diberikan oleh solusi SSL-AES pada GPU adalah sekitar 7,8 kali dibandingkan dengan implementasi CPU pada OpenSSL.

REFERENSI

[1] A. Hodjat, I. Verbauwhede. (2004). *Minimum Area Cost for a 30 to 70 Gbits/s AES Processor*. IEEE Computer Society Annual Symposium on VLSI 2004, Eerging Trends in VLSI System Design.

[2] Chandra, P., Messier, M., & Viega, J. (2002). *Network Security with OpenSSL*. Sebastopol, CA: O'Reilly.

[3] Daemen, Joan & Rijmen, Vincent. (2001). *The Design of Rijndael: The Advanced Encrpytion Standard*. Berlin: Springer.

[4] Harrison, Owen & Waldron, John. (2007). *AES Encryption Implementation and Analysis on Commodity Graphics Processing Units*. The 9th international workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer.

[5] Liu, Gu et. al. (2009). *A Program Behavior Study of Block Cryptography Algorithms on GPGPU*. 2009 International Conference on Frontier of Computer Science and Technology.

[6] National Institute of Standard and Technology. (2001). *Federal Information Processing Standards Publication 197, Announcing the Advanced Encryption Standard (AES)*. Gaithersburg, MD: National Institute of Standard and Technology.

[7] NVIDIA Corporation. (2010). *NVIDIA CUDA C Programming Guide*. Santa Clara, CA: NVIDIA Corporation.

[8] Manavski, Svetlin A. (2007). *CUDA Compatible GPU as an Efficient Hardware Accelerators for AES Cryptography*. IEEE International Conference on Signal Processing and Communications. New York, NY: IEEE.

[9] OpenSSL. http://www.openssl.org/. Diakses 5 Mei 2011.

[10] She, Cuai et. al. (2008). A performance study of general-purpose applications on graphics processors using CUDA.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Mei 2011

Adityo Jiwandono, 13507015