Paralelisasi Algoritma Pencocokan String Knuth-Morris-Pratt dengan NVIDIA CUDA pada Network Intrusion Detection System

Ramadhan Satya Putra, Waskitho Wibisono, dan Baskoro Adi Pratomo
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: waswib@if.its.ac.id

***Abstrak*— Keberadaan teknologi informasi yang terus berkembang dengan pesat menjadikan kebutuhan akan penggunaanya semakin hari semakin meningkat. Dewasa ini penggunaan internet untuk berbagai macam kebutuhan sangat sering dijumpai.. Tentu dengan berbagai macam penggunaan internet tersebut dibutuhkan metode untuk mengamankan jaringannya.**

Pada tugas akhir ini menerapkan konsep NIDS (Network Intrusion Detection System). NIDS adalah salah satu dari tiga jenis IDS. NIDS pada tugas akhirini akan memanfaatkan GPGPU pada proses string matching. Proses string matching menggunakan algoritma KMP dan berjalan pada GPU. Tujuan penggunaan GPU adalah peningkatan performa dalam segi kecepatan proses pencocokan string dan pengurangan beban terhadap CPU.

Hasil evaluasi dari tugas akhir ini menunjukkan bahwa dengan memanfaatkan GPGPU runtime dari aplikasi lebih lama. Sedangkan untuk penggunaan CPU lebih stabil dalam angka 23%-24.5%.

Kata Kunci*: Algoritma KMP, CPU, GPU, *GPGPU*, *NIDS*, *String Matching

I. PENDAHULUAN

Keberadaan teknologi informasi yang terus berkembang dengan pesat menjadikan kebutuhan akan penggunaanya semakin hari semakin meningkat. Transaksi data melalui internet telah menjadi kebutuhan wajib hampir dari semua perangkat lunak yang ada saat ini. Perangkat lunak seperti media sosial, *cloud server*, *online game*, aplikasi layanan pemerintah, dan aplikasi pengontrol jarak jauh. Tentu dengan berbagai macam penggunaan internet tersebut dibutuhkan metode untuk mengamankan jaringannya. Sistem pendeteksi serangan atau yang pada umumnya disebut IDS (*Intrusion Detection System*) [4] merupakan senjata utama untuk mengamankan suatu jaringan di mana sistem ini nantinya bertugas untuk mengidentifikasi dan mencatat apakah suatu paket merupakan bentuk serangan atau paket biasa.

Cara kerja IDS sendiri adalah *sniffing*, pencocokan paket, dan *logging* [4]. Pencocokan tidak hanya dilakukan pada paket tertentu, semua paket yang melalui jaringan dan dilayani akan diperiksa apakah paket tersebut berupa paket normal atau paket yang mengandung elemen sesuai dengan elemen *rule* yang ditetapkan. Berkembangnya *traffic* internet dewasa ini mencapai angka puluhan Gbps (*Gigabit per second*) dan bukan tidak mungkin akan mencapai ratusan atau bahkan ribuan Gbps nantinya.

Kemampuan IDS pada umumnya bergantung pada kemampuan CPU (*Central Processing Unit*) . Pada umumnya CPU bukan hanya untuk menjalankan IDS saja namun untuk mengatur jalannya seluruh sistem komputer sehingga tidak jarang suatu CPU kewalahan untuk mengatur IDS dikarenakan banyaknya jumlah data atau paket yang harus diproses. Beban berat CPU ketika menjalankan fungsi IDS ini dapat dikurangi dengan memanfaatkan bantuan GPU (*Graphic Processing Unit*).

GPU adalah inti dari kartu grafis. Kartu grafis pada umumnya dirancang untuk keperluan *graphic rendering* seperti pada aplikasi editor gambar, video, dan *game*. Seiring berjalannya waktu, produsen kartu grafis ternama seperti NVDIA dan ATI mengembangkan kemampuan inti dari kartu grafis untuk melakukan GPGPU (*General Purpose on Graphic Processing Unit*) [7], yaitu kemampuan melakukan komputasi secara umum layaknya sebuah CPU. NVIDIA mengembangkan CUDA [6] untuk memenuhi kebutuhan akan GPGPU. Kelebihan dari GPU ini adalah memiliki banyak inti daripada CPU sehingga dapat dimanfaatkan untuk melakukan paralelisasi terhadap proses yang sedang dijalankan.

Deteksi serangan pada suatu jaringan yang memilki *data traffic* besar sudah pasti membutuhkan kemampuan suatu sistem untuk mencocokkan banyaknya paket yang ada dengan *rule* yang sudah tersimpan pada aplikasi. Dengan memanfaatkan bantuan GPU, proses pencocokan *rule* tersebut dapat dilakukan dengan mudah dan relatif lebih cepat daripada memanfaatkan CPU saja, akan tetapi proses pendeteksian serangan ini tidak dapat berjalan dengan serta merta pada GPU saja namun tetap memerlukan CPU karena secara *utility* jenis komputasi yang didukung oleh CPU lebih banyak daripada GPU. Seperti contoh proses penangkapan paket dan pembuatan *log* harus dilakukan pada CPU. Kekurangan lain dari GPU adalah bertambahnya beban listrik yang dibutuhkan.

Tugas Akhir ini bertujuan untuk memanfaatkan kemampuan CPU dan GPU secara optimal pada studi kasus NIDS, optimal yang dimaksud adalah dapat mempercepat proses komputasi dengan memanfaatkan GPU. Dengan adanya aplikasi pendeteksi serangan ini, diharapkan kedepannya akan membantu mengamankan suatu jaringan yang memiliki *data traffic* besar tanpa memerlukan sebuah komputer yang memiliki spesifikasi sangat tinggi, cukup hanya dibekali dengan GPU, yang mendukung GPGPU, sebagai tambahan untuk proses pencocokan `string`.

II. URAIAN PENELITIAN

A. NIDS

NIDS [4] (*Network Intrusion Detection System*) adalah salah satu tipe dari IDS (*Intrusion Detection System*) yang berfungsi untuk menganalisa hasil dari tangkapan paket pada jaringan. Analisa yang dilakukan adalah pencocokan antara konten di dalam paket dan konten pada *rule*. hasil dari analisa paket tersebut dapat berupa peringatan, pengabaian, dan pencatatan.

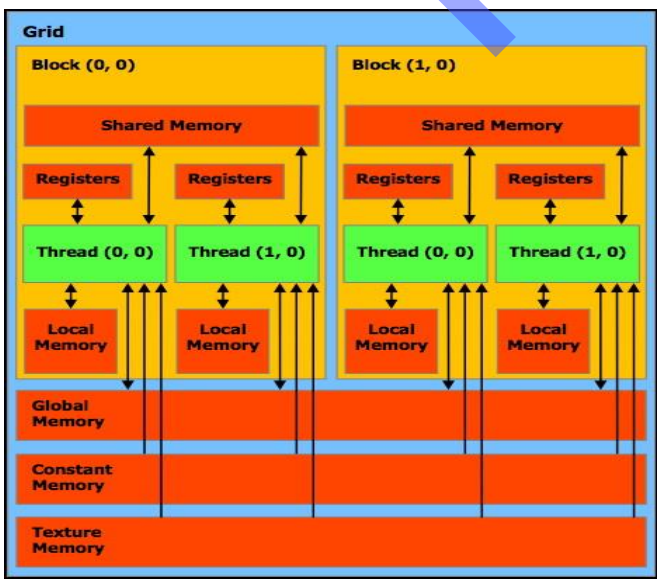
B. Jpcap

Jpcap [3] adalah kumpulan Java *class* yang menyediakan *interface* dan sistem untuk melakukan *packet capture* pada jaringan. Tugas Akhir ini menggunakan Jpcap untuk mendapatkan data dari jaringan yang nantinya akan menjadi masukan untuk kemudian diolah. Salah satu kegunaan Jpcap yang sering dijumpai adalah *offline capture*. Pengertian dari *offline capture* adalah pembacaan *dump file* yang didapatkan dari hasil aktual suatu *data traffic*. *Offline capture* dilakukan untuk menganalisa *data traffic* yang sudah disimpan untuk tujuan tertentu antara lain, menganalisa kebiasaan pengguna, mendeteksi anomali yang mungkin terlewat dari pengawasan, dan lain sebagainya .

C. NVIDIA CUDA

NVIDIA CUDA [6] adalah platform komputasi paralel dan model pemrograman yang memungkinkan peningkatan secara signifikan pada performa komputasi dengan cara memanfaatkan GPU (Graphic Processing Unit) sebagai sumber daya. Secara umum metode ini disebut GPGPU (General Purpose Computing on Graphic Processing Unit).

Beberapa *thread* tersebut terkumpul dalam satu *block* yang memiliki *shared memory* yang hanya bisa diakses oleh *thread* yang terdapat dalam satu *block* tersebut. Kumpulan-kumpulan *block* tersebut membentuk sebuah *grid*. Selain *global* dan *shared memory*, terdapat *constant* dan *texture memory* yang bisa diakses oleh setiap *thread* pada CUDA dan juga *local memory* dan *registers* yang hanya setiap *thread* itu sendiri yang bisa mengakses.



Gambar. 1. Arsitektur CUDA [6]

D. Algoritma Knuth-Morris-Pratt

Algoritma Knuth Morris Pratt [2] adalah algoritma pencocokkan *string*, algoritma ini sering disebut sebagai Algoritma KMP. Meskipun nama algoritmanya adalah Knuth-Morris-Pratt pada kenyataannya algoritma ini dikembangkan secara terpisah. Pertama dikembangkan oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun publikasinya dilakukan secara bersamaan pada tahun 1977.

```
prosedur HitungPinggiran (input m :
integer,
P : string)
kamus :
k, g : integer
algoritma :
b[] = array[1..m] of integer
b[0] = 0
q = 1
k = 0
for q ← 1 to m-1 do
        while ((k > 0) and (P[q] ≠ P[k]))
do
                k ←b[k]
        endwhile
        if P[q] = P[k] then
                k ← k+1
        endif
        b[q] ← k
endfor
```

(a)

```
function KMPSearch ( input m : integer,
n : integer, P : string, T string) →
boolean
( mengembalikan true jika pattern P
ditemukan
dalam teks T)
kamus:
i, j : integer
ketemu : boolean
prosedur HitungPinggiran ( input m :
integer,
P : string)
algoritma
HitungPinggiran(m, P)
i 0
j 0
ketemu = false
while ((j > 0) and (P[j] T[i[]) do
        j ← b[j-1]
endwhile
if P[j] = T[i] then
        j ← j+1
endif
if j = m then
        ketemu ← true
else
        i ← i+1
endif
return ketemu
```

(b)

Gambar. 2. (a) *Pseudocode* perhitungan fungsi pinggiran (b) *pseudocode* algrotima KMP

Cara kerja Algoritma KMP adalah perhitungan fungsi pinggiran untuk menentukan jumlah *skip* pada proses pencocokan. Fungsi pinggiran adalah perhitungan nilai *prefix* dan *sufix* dari *string pattern* yang akan dicocokan denga *string text*. Penerapan konsep *brute force* pada pencocokan *string* adalah mencocokan dua buah *string* secara berurutan dari karakter pertama hingga karakter terakhir. Algoritma KMP juga melakukan pencocokan seperti demikian namun dengan bantuan fungsi pinggiran dapat ditentukan berapa karakter yang harus dilewati bila terjadi ketidakcocokan.

E. Snort Rule

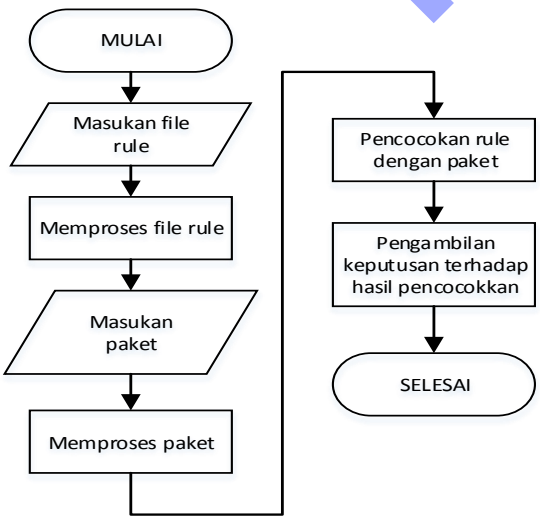
Snort *rule* [1] adalah *rule* yang digunakan pada aplikasi NIDS bernama Snort. Isi dari Snort *rules* adalah *action*, alamat IP asal, alamat IP tujuan, *port* asal, *port* tujuan, dan *option* yang akan dicocokkan. Kolom *action* berisi perintah apa yang akan dilakukan ketika kondisi *rule* terpenuhi. Kolom *option* berisi opsi lain yang dibutuhkan seperti *content* yang berisi konten apa yang akan dicocokkan dan *message* yang merupakan isi dari pesan yang akan diberikan ke pengguna.

F. JCuda

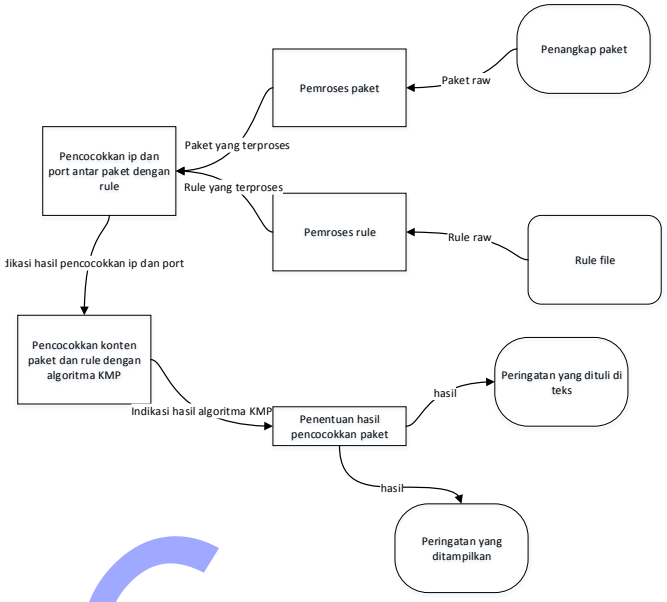
JCuda [5] adalah *binding* pada bahasa pemrograman Java untuk CUDA. Syarat menggunakan *library* ini adalah GPU (*Graphic Processing Unit*) NVIDIA yang mendukung CUDA dan Cuda Toolkit yang sudah terpasang pada komputer yang akan digunakan. Penggunaan JCuda membuat pemanggilan fungsi *kernel* yang menggunakan bahasa pemrograman C/C++ dapat dipanggil dari Java *class*.

III. PERANCANGAN DAN IMPLEMENTASI

Alur Kerja aplikasi secara umum terdapat pada Gambar 3. Gambar 4 berisi diagram alir data dari aplikasi. Aplikasi pertama tama akan menerima dan memproses *rule* dan paket sebelum kemudia dicocokkan. Proses pencocokkan berjalan pada GPU NVIDIA. Setelah proses pencocokkan selesai aplikasi akan mengambil keputusan untuk keluaran sesuai dengan hasil dari pencocokkan.



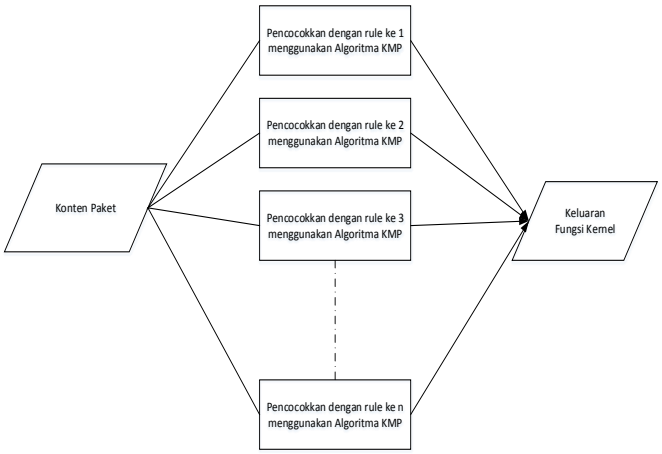
Gambar. 3. Alur Kerja aplikasi



Gambar. 4. Diagram alir data

Aplikasi ini dibangun dengan Bahasa Pemrograman Java untuk fungsi utama dan Bahasa Pemrograman C untuk fungsi kernel atau fungsi yang berjalan pada GPU. JCuda dimanfaatkan untuk mengakses fungsi kernel dari fungsi utama. Jpcap digunakan untuk menangkap paket yang kemudian diproses sesuai dengan kebutuhan aplikasi.

Algoritma KMP sebagai pencocok string berada di dalam fungsi kernel. Konsep pembagian *thread* terdapat pada Gambar 5. Pembagian yang dilakukan adalah pembagian *thread* sesuai dengan jumlah *rule*. Setiap *thread* akan mencocokkan konten pada paket sesuai dengan konten pada *rule* yang ditangani. Berikutnya fungsi *kernel* akan mengeluarkan keluaran dalam bentuk integer yang berikutnya ditampung dalam bentuk integer juga oleh aplikasi utama. Keluaran yang berupa integer tersebut akan dijadikan patokan oleh aplikasi untuk menentukan keputusan apa yang akan diambil oleh aplikasi diantara *alert*, *log*, atau *pass*.



Gambar. 5. Konsep pembagian *thread* yang dterapkan pada aplikasi

IV. UJI COBA

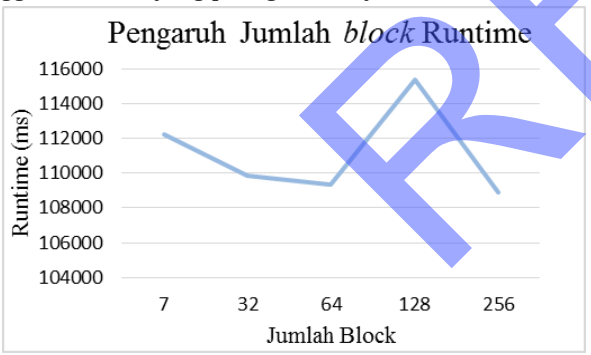
Uji coba yang dilakukan aplikasi ini menggunakan paket yang berasal dari 1999 DARPA *dataset* [9]. Penggunaan 1999 DARPA *Dataset* dipilih karena *dataset* tersebut sudah teruji terbukti dengan digunakannya *dataset* tersebut pada berbagai publikasi ilimiah. *Dataset* tersebut juga mengandung berbagai macam bentuk serangan. Snort *rule* yang digunakan adalah *rule* mengenai Dos dan Imap [8] karena kedua bentuk serangan tersebut terdapat pada 1999 DARPA *dataset*.

A. Uji Coba Performa Berdasarkan Alokasi Block

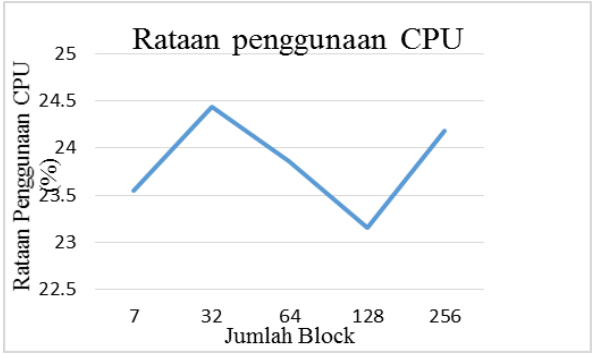
Uji coba ini difokuskan untuk mengetahui performa dari aplikasi sesuai dengan jumlah *block* yang berubah – ubah. Ketika berjalan aplikasi tetap menggunakan jumlah *block* sejumlah *rule* yang digunakan namun terbukti pada Gambar 6 bahwa perbedaan jumlah *block* mempengaruhi performa aplikasi.

B. Uji Coba Performa Berdasarkan Jumlah Paket

Uji coba ini difokuskan untuk mengetahui performa aplikasi dalam segi penggunaan CPU dan *runtime* dengan jumlah paket sebagai pembanding. Uji coba akan menggunakan *dataset* yang berisi 500, 1000, 2000, dan 3000 paket dari 1999 DARPA *dataset week 2 Monday* yang sudah diproses sebelumnya dan *rule* yang berisi modifikasi dari Snort *rule* untuk dos dan imap yang disimpan dengan nama `dos-imap-modified-82.rules` jumlah *rule* yang terdapat pada *file* tersebut adalah 82. Konfigurasi di dalam aplikasi yang dilakukan adalah alokasi *block* yaitu sejumlah 128 karena sesuai dengan uji coba performa terhadap *block* yang menunjukan bahwa dengan *block* berjumlah 128 terjadi penggunaan CPU yang paling rendah yaitu 23.16%.

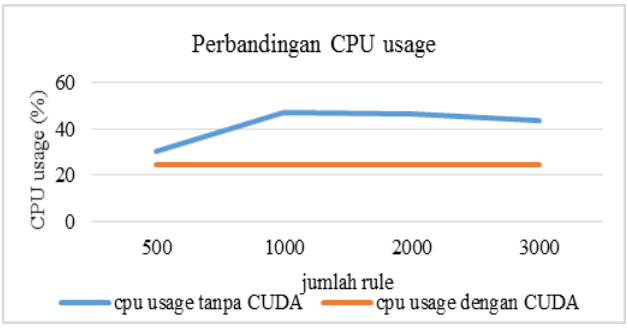


(a)

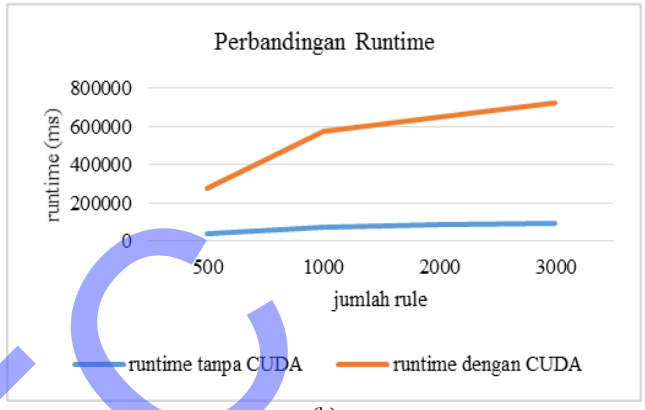


(b)

Gambar. 6. (a) Grafik perbadingan jumlah *block* terhadap *runtime* (b) Grafik perbandingan jumlah *block* terhadap penggunaan CPU



(a)



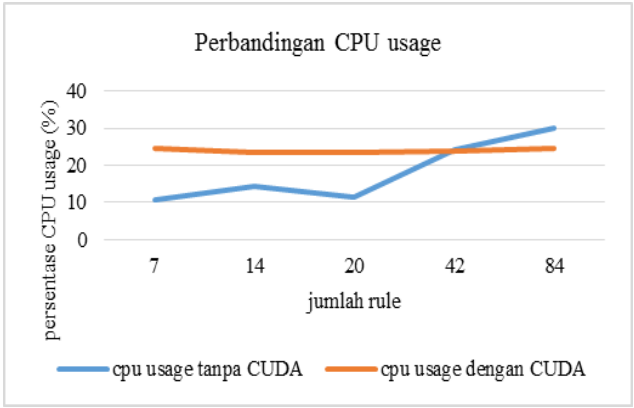
(b)

Gambar. 7. (a) Grafik perbadingan jumlah paket terhadap *runtime* (b) Grafik perbandingan jumlah paket terhadap penggunaan CPU

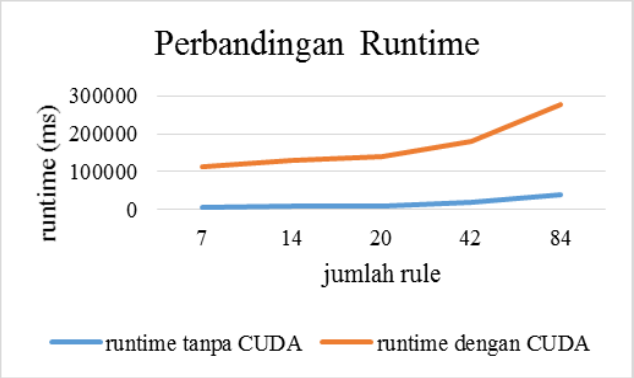
Sebagai pembanding performa aplikasi dibandingkan dengan performa aplikasi yang tidak menggunakan CUDA atau fungsi kernel di dalamnya. Perbandingan performa dapat dilihat pada Gambar 7.

C. Uji Coba Performa Berdasarkan Jumlah Rule

Uji coba performa terakhir yang dilakukan adalah uji coba dengan jumlah *rule* sebagai pembanding. Uji coba akan menggunakan *dataset* yang berisi 500 paket hasil olahan dari 1999 DARPA *dataset week 2 Monday*. *Rule* yang digunakan merupakan hasil modifikasi Snort *rule* untuk dos dan imap. Modifikasi yang dilakukan antara lain perubahan IP asal, port asal, IP tujuan dan port tujuan. Modifikasi lain yang dilakukan adalah penambahan jumlah *rule*. Jumlah *rule* yang digunakan adalah 7, 14, 20, 42, dan 84.



Gambar. 8. Grafik perbadingan jumlah *rule* terhadap penggunaan CPU



Gambar. 8. Grafik perbadingan jumlah *rule* terhadap *runtime*

Sama seperti uji coba sebelumnya, performa aplikasi dibandingkan dengan performa aplikasi yang tidak menggunakan CUDA. Perbandingan performa terdapat pada Gambar 8 untuk grafik perbadingan jumlah *rule* terhadap penggunaan CPU dan Gambar 9 untuk grafik perbadingan jumlah *rule* terhadap *runtime*.

V. KESIMPULAN

1. Algoritma KMP dapat diimplementasikan pada GPGPU menggunakan *platform* NVIDIA CUDA yang berjalan pada perangkat VGA NVIDIA tipe GeForce GTX 750 Ti.
2. Masukan dan keluaran dari fungsi *kernel* dapat disesuaikan dengan kebutuhan aplikasi, yaitu variabel dengan tipe `integer`, `string` untuk menyimpan konten paket, `array of string` untuk menyimpan konten *rule*, dan `array of integer` untuk menyimpan keluaran fungsi kernel.
3. Konsep paralel dengan NVIDIA CUDA dapat diterapkan pada studi kasus *Network Intrusion Detection System*. Hasil keluaran yang selalu konsisten membuktikan bahwa manajemen *thread* dapat disesuaikan dengan kebutuhan aplikasi.
4. Pada studi kasus NIDS, alokasi *block* 128 menghasilkan penggunaan CPU yang paling rendah yaitu 23.16% daripada alokasi *block* lainnya. Alokasi *block* 256 menghasilkan *runtime* yang paling rendah yaitu 108926 ms.
5. Aplikasi dengan CUDA memiliki rata rata CPU *usage* yang stabil yaitu antara 23.5% - 24.3%, dibandingankan aplikasi tanpa CUDA yang CPU *usage* – nya semakin meningkat bila jumlah *rule* atau paket semakin banyak, yaitu 10.81% - 47.29%.
6. Aplikasi tanpa CUDA memiliki *runtime* yang lebih sedikit. Rata – rata *runtime* aplikasi tanpa CUDA mencapai 91.7% lebih cepat daripada aplikasi dengan CUDA untuk pembanding jumlah *rule*. Aplikasi tanpa CUDA juga 86.6% lebih cepat daripada aplikasi dengan CUDA untuk pembanding jumlah paket.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih sebesar – besarnya kepada Allah SWT, kedua orang tua penulis Prijono Satyabhakti dan Irma Monoarfa, Rosita Mekayanti kakak penulis, Pak Waskitho dan Pak Baskoro dosen pembimbing, bapak/ibu dosen dan karyawan Jurusan Teknik Informatika, sahabat dan kerabat dekat, serta berbagai pihak yang telah membantu untuk tercapainya tujuan dari tugas akhir ini.

DAFTAR PUSTAKA

[1] O'Reilly, "Write Your Own Snort Rules," [Online]. Available: http://oreilly.com/pub/h/1393. [Accessed 2014 June 20].

[2] D. Knuth, J. H. j. Morris and V. Pratt, "Fast Pattern Matching in Strings," *SIAM Journal on Computing*, pp. 323-350, 1977.

[3] pcharles, "jpcap network capture library," jpcap, [Online]. Available: http://jpcap.sourceforge.net/. [Accessed 8 June 2014].

[4] SANS Institute, "Understanding Intrusion Detection System," *SANS Institute Reading Room*, p. 9, 2001.

[5] jcuda, "jcuda tutorial," jcuda, [Online]. Available: http://jcuda.org/tutorial/TutorialIndex.html. [Accessed 8 June 2014].

[6] NVIDIA, "What is CUDA | NVIDIA Developer Zone," [Online]. Available: https://developer.nvidia.com/what-cuda. [Accessed 3 March 2014].

[7] NVIDIA, "What is GPU Computing?," [Online]. Available: http://www.nvidia.com/object/what-is-gpu-computing.html. [Accessed 11 November 2014].

[8] S. Paliwal and R. Gupta, "Denial-of-Service, Probing & Remote to User (R2L) Detection using Genetic Algorithm," *International Journal of Computer Application*, vol. 60, no. December 2012, pp. 57-62, 2010.

[9] MIT Lincoln Laboratory, "MIT Lincoln Laboratory: Cyber system & technology: DARPA Intrusion Detection," MIT Lincoln Laboratory, [Online]. Available: http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/docs/index.html. [Accessed 30 November 2014].