IMPLEMENTASI FILTER SPASIAL LINEAR PADA VIDEO *STREAM* MENGGUNAKAN *FPGA HARDWARE ACCELERATOR*

SULAEMAN
Universitas Hasanuddin
sulaeman16h@student.unhas.ac.id

Abstract

Berbagai macam akselerator telah dikembangkan dalam meningkatkan kinerja dan efisiensi energi untuk menangani komputasi berat, salah satu diantaranya yaitu FPGA. FPGA mampu menangani beban komputasi yang begitu berat sehingga dapat digunakan untuk Digital Signal Processing, Image Processing, Neural Network, dan sebagainya. Pada penelitian ini penulis mencoba mengkaji kinerja yang dimiliki ARM prosesor dan FPGA pada FPGA Development Board Xilinx PYNQ Z2 dalam penerapan filter spasial linear pada video stream. Kernel filter yang digunakan pada penelitian ini yaitu average blur, gaussian blur, laplacian, sharpen, sobel horizontal dan sobel vertical. Parameter yang digunakan untuk mengukur kinerja ARM prosesor dan FPGA yaitu waktu komputasi, frame rate (FPS), penggunaan CPU, penggunaan memory, resident memory (RES), shared memory (SHR) dan virtual memory (VIRT). Rata-rata waktu komputasi yang dibutuhkan untuk menerapkan filter spasial linear pada 200 frame dengan ARM prosesor adalah 29.06 detik sedangkan dengan FPGA rata-rata hanya dibutuhkan 3.32 detik. Waktu komputasi dengan FPGA 88.85% lebih baik dibandingkan dengan ARM prosesor. Video hasil filter dengan ARM prosesor memperoleh rata-rata 6.95 fps sedangkan dengan FPGA rata-rata 60.37 fps. FPS dengan FPGA 88.49% lebih baik lebih baik dibandingkan ARM prosesor. Penggunaan CPU pada FPGA 14.89% lebih baik, penggunaan memory pada FPGA 2.02% lebih baik, penggunaan resident memory 2.07% lebih baik, dan penggunaan shared memory 4.08% lebih baik dibandingkan dengan ARM prosesor. Sedangkan penggunaan virtual memory pada ARM prosesor 0.03% lebih baik dibandingkan FPGA.

Kata Kunci : *filter spasial linear, FPGA, ARM prosesor, video stream, video processing*

1. PENDAHULUAN

Citra digital merupakan citra yang dihasilkan dari pengolahan secara digital dengan merepresentasikan citra secara numerik dengan nilai-nilai diskrit. Suatu citra digital dapat direpresentasikan dalam bentuk matriks dengan fungsi *f(x,y)* yang terdiri dari M kolom dan N baris. Perpotongan antara baris dan kolom disebut sebagai piksel [5].

Pengolahan citra digital merupakan proses mengolah piksel di dalam citra secara digital untuk tujuan tertentu. Berdasarkan tingkat pemrosesannya pengolahan citra digital dikelompokkan menjadi tiga kategori, yaitu: *low-level*, *mid-level* dan pemrosesan *high-level*. Pemrosesan *low-level* dilakukan dengan operasi primitif seperti *image preprocessing* untuk mengurangi derau (*noise*), memperbaiki kontras citra dan mempertajam citra (*sharpening*). Pemrosesan *mid-level* melibatkan tugas-tugas seperti segmentasi atau mempartisi gambar menjadi beberapa bagian atau objek, deskripsi objek untuk dilakukan pemrosesan lanjutan, dan klasifikasi objek yang terdapat dalam citra digital. Pemrosesan *high-level* merupakan proses tingkat lanjut dari dua proses sebelumnya, dilakukan untuk mendapat informasi lebih yang terkandung dalam citra seperti *pattern recognition*, *template matching*, *image analysis* dan sebagainya [5].

Konsep filter spasial pada pengolahan citra digital berasal dari penerapan transformasi Fourier untuk pemrosesan sinyal pada domain frekuensi. Istilah filter spasial ini digunakan untuk membedakan proses ini dengan filter pada domain frekuensi. Proses filter dilakukan dengan cara menggeser filter kernel dari titik ke titik dalam citra digital.

Untuk meningkatkan kinerja dan efisiensi energi dari sebuah program, berbagai jenis akselerator telah dikembangkan, salah satu diantaranya yaitu FPGA [4]. *Field Programmable Gate Arrays* atau FPGA adalah perangkat semikonduktor yang berbasis *matriks configurable logic block* (CLBs) yang terhubung melalui interkoneksi yang dapat diprogram. FPGA dapat diprogram ulang dengan aplikasi atau fungsi yang diinginkan setelah *manufacturing*. Fitur ini yang membedakan FPGA dengan *Application Specific Integrated Circuits* (ASICs), yang dibuat khusus untuk tugas tertentu saja [17].

FPGA telah menunjukkan kinerja yang sangat tinggi di dalam banyak aplikasi dalam pemrosesan citra.

Namun CPU dan CPU terbaru memiliki potensi kinerja tinggi untuk masalah-masalah tersebut. CPU terbaru mendukung *multi-core*, dimana masing-masing *core* mendukung SIMD (*Single Instruction, Multiple Data*) yang telah dikembangkan dan dijalankan hingga 16 operasi pada 128 bit data dalam satu *clock cycle*. GPU terbaru mendukung sejumlah besar *core* yang berjalan secara paralel, dan kinerja puncaknya mengungguli CPU [1].

Paralelisme dalam SIMD pada CPU terbatas, tetapi frekuensi operasional CPU sangatlah tinggi, dan CPU diharapkan dapat menunjukkan kinerja yang tinggi dalam aplikasi yang dimana *cache memory* berjalan dengan baik. Ukuran *cache memory* cukup besar untuk menyimpan seluruh citra di banyak aplikasi pemrosesan citra, dan CPU dapat menjalankan algoritma yang sama dengan FPGA meskipun *bandwith memory* yang dibutuhkan tinggi [1].

Frekuensi operasional GPU lebih cepat dibandingkan dengan FPGA, namun sedikit lebih lambat dibandingkan dengan CPU. Akan tetapi, GPU mendukung banyak *core* yang berjalan secara paralel sehingga kinerja puncaknya mengungguli CPU. Namun *core*-nya dikelompokkan, dan transfer data antara kelompok sangatlah lambat. Selain itu, ukuran *local memory* yang disediakan masing-masing kelompok sangat kecil. Karena keterbatasan ini, GPU tidak dapat menjalankan algoritma yang sama seperti FPGA dalam beberapa masalah aplikasi [1].

Pada FPGA terdahulu tidak terdapat prosesor untuk menjalankan *software* apapun, sehingga ketika ingin mengimplementasikan aplikasi haruslah merancang sirkuit dari awal. Sebagian besar FPGA sekarang telah dirangkai dengan prosesor dalam satu *board*, sering disebut sebagai FPGA Development Board. Xilinx PYNQ-Z2 dibangun dari prosesor ARM Cortex-A9, sehingga dapat menjalankan beberapa *software* seperti *python* tanpa harus merancang sirkuitnya dari awal. Akan tetapi, kinerja yang dimiliki oleh prosesor ARM pada FPGA Development Board tentu berbeda dengan kinerja fungsi arsitektur FPGA itu sendiri sehingga dapat dikaji lebih dalam mengenai perbandingan kinerja dari keduanya.

2. Landasan Teori

2.1. Citra Digital

Citra digital dapat didefinisikan sebagai fungsi *f(x,y)* berukuran M baris dan N kolom, dengan *x* dan *y* adalah koordinat spasial, dan amplitudo *f* di titik koordinat (x,y) dinamakan intensitas atau tingkat keabuan dari citra pada citra tersebut [10]. Pada umumnya warna dasar dalam citra RGB menggunakan penyimpanan 8 bit untuk menyimpan data warna, yang berarti setiap warna mempunyai gradasi sebanyak 255 warna . Dewasa ini, citra digital dapat menggunakan 16 bit untuk menyimpan data warna dasarnya, hal ini menyebabkan semakin banyak gradasi warnanya sehingga citra yang dihasilkan memiliki tingkat warna yang jauh lebih banyak. Namun tentu saja hal ini mengakibatkan ukuran file citra digital yang dihasilkan juga menjadi semakin besar walaupun dengan ukuran yang sama. Berdasarkan jenis warnanya citra digital dibagi menjadi 3 jenis yaitu citra biner, citra grayscale dan citra warna.

2.1.1 Citra Biner (monokrom)

Citra biner adalah citra yang hanya memiliki dua warna saja yaitu hitam dan putih. Warna hitam direpresentasikan dengan 1 dan warna putih direpresentasikan dengan 0. Dibutuhkan 1 bit di memori untuk menyimpan nilai warna pada setiap piksel citra biner. Contoh citra biner dapat dilihat pada gambar 1(a).

2.1.2 Citra Grayscale

Banyaknya warna pada citra *grayscale* tergantung pada jumlah bit yang disediakan di memori untuk menampung kebutuhan warna pada citra ini. Citra *grayscale* yang 2 bit memiliki 4 gradasi warna, citra *grayscale* 3 bit memiliki 8 gradasi warna, citra *grayscale* dengan 8 bit memiliki 256 gradasi warna dan seterusnya. Semakin besar jumlah bit warna yang disediakan di memori, semakin banyak dan semakin halus gradasi warna yang terbentuk. Pada umumnya citra digital *grayscale* menggunakan 8 bit memori dengan derajat keabuan dari 0 sampai 255. Contoh citra *grayscale* dapat dilihat pada gambar 1(b).

2.1.3 Citra Warna

Setiap piksel pada citra warna mewakili warna yang merupakan kombinasi dari tiga warna dasar (RGB = red, green, blue). Pada umumnya setiap warna dasar menggunakan penyimpanan 8 bit, yang berarti setiap warna mempunyai gradasi sebanyak 255 warna. Berarti setiap piksel mempunyai kombinasi warna sebanyak 255 x 255 x 255 =16 juta warna lebih. Contoh citra warna dapat dilihat pada gambar 1(c).

Gambar 1: *(a) Contoh citra biner, (b) contoh citra grayscale, (c) contoh citra warna.*

2.2. Pengolahan Citra Digital

Pengolahan citra digital merupakan proses mengolah piksel-piksel di dalam citra secara digital untuk tujuan tertentu. Berdasarkan tingkat pemrosesannya pengolahan citra digital dikelompokkan menjadi tiga kategori, yaitu: *low-level*, *mid-level* dan pemrosesan *high-level*. Pemrosesan *low-level* dilakukan dengan operasi primitif seperti *image preprocessing* untuk mengurangi derau (*noise*), memperbaiki kontras citra dan mempertajam citra (*sharpening*). Karakteristik dari pemrosesan *low-level* yaitu keluaran atau hasil dari pemrosesannya berupa citra digital. Pemrosesan *mid-level* melibatkan tugas-tugas seperti segmentasi (mempartisi gambar menjadi beberapa bagian atau objek), deskripsi objek untuk dilakukan pemrosesan lanjutan, dan klasifikasi objek dalam citra digital. Karakteristik dari pemrosesan *mid-level* yaitu keluaran atau hasilnya berupa atribut atau fitur seperti, kontur, tepi, atau objek yang terdapat dalam citra tersebut. Pemrosesan *high-level* merupakan proses tingkat lanjut dari dua proses sebelumnya, dilakukan untuk mendapat informasi lebih yang terkandung dalam citra [5].

Berdasarkan tujuannya pengolahan citra juga dapat dibagi menjadi beberapa bagian yaitu: *image enhancement*, *image restoration*, *image analysis* dan *image compression* [14].

2.2.1 Image Enhancement

Image Enhancement adalah metode pengolahan citra digital untuk membuat citra tampak lebih baik atau dilakukan peningkatan untuk analisis tertentu. Namun hal ini dapat menyebabkan pengorbanan aspek lain dari citra tersebut. Penerapan filter, penghalusan citra, memperbaiki kontras dan morfologi citra adalah contoh *image enhancement*.

2.2.2 Image Restoration

Image restoration adalah metode pengolahan citra untuk memulihkan citra dari penurunan kualitas atau citra yang rusak karena derau (*noise*). Pada dasarnya metode ini berbeda dengan *image enhancement* yang berkaitan dengan ekstraksi fitur pada citra.

2.2.3 Image Analysis

Image analysis adalah metode pengolahan citra untuk menghitung besaran kuantitatif dari citra untuk menghasilkan deskripsinya. Metode ini dilakukan dengan mengekstraksi ciri-ciri tertentu yang membantu dalam identifikasi objek [14].

2.2.4 Image Compression

Metode ini dilakukan agar citra dapat direpresentasikan dalam bentuk yang lebih kompak sehingga memerlukan memori yang lebih sedikit. Metode ini dapat dilakukan dengan mengurangi redundansi dari data-data yang terdapat dalam citra sehingga dapat disimpan atau ditransmisikan secara efisien.

2.3. Filter Spasial

Konsep filter spasial pada pengolahan citra digital berasal dari penerapan transformasi Fourier untuk pemrosesan sinyal pada domain frekuensi. Istilah filter spasial ini digunakan untuk membedakan proses ini dengan filter pada domain frekuensi. Proses filter dilakukan dengan cara menggeser filter kernel dari titik ke titik dalam citra digital. Istilah *mask*, *kernel*, *template*, dan *window* merupakan istilah yang sama dan sering digunakan dalam pengolahan citra digital [5]. Dalam penelitian ini peneliti menggunakan istilah kernel untuk istilah tersebut.

Proses filter dalam pengolahan citra digital dilakukan dengan memanipulasi sebuah citra menggunakan kernel untuk menghasilkan citra yang baru, sehingga dengan kernel yang berbeda maka citra hasil yang didapat juga akan berbeda.

2.3.1 Operator Linear dan Non-linear

Didefinisikan H sebuah operator dengan *input* dan *output* adalah citra digital. H dikatakan operator linear jika untuk untuk sembarang gambar f dan g, dan untuk sembarang skalar a dan b berkalu,

$$H(af + bg) = aH(f) + bH(g) \tag{1}$$

Dengan kata lain hasil dari operator linear dengan jumlahan dua buah citra (yang telah dikali dengan konstanta a dan b) identik dengan hasil operator linear pada masing-masing gambar, dikali dengan konstanta yang sama, kemudian hasilnya dijumlahkan.

Sebagai contoh, sebuah operator dengan fungsi yang menjumlahkan K citra adalah operator linear. Operator yang menghitung nilai mutlak dari perbedaan dua gambar adalah tidak linear. Operator yang tidak memenuhi persamaan (1) dikatakan non-linear [5].

2.4. Kernel

2.4.1 Average Blur

Average blur atau biasa juga disebut *box filter* adalah salah satu filter yang digunakan untuk menghaluskan citra dan mengurangi derau. Secara sederhana nilai sebuah piksel yang baru adalah nilai rata-rata dari nilai piksel tersebut dengan nilai piksel tetangganya [8]. Berikut kernel *Average blur* yang digunakan dalam penelitian ini:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{2}$$

2.4.2 Gaussian Blur

Filter ini juga digunakan untuk menghaluskan citra dan mengurangi derau. Idenya mirip seperti *Average blur*, nilai piksel yang baru dibentuk dari nilai piksel tetangganya, tetapi dengan memberikan bobot yang lebih kuat pada nilai pikselnya sendiri diikuti dengan bobot yang lebih rendah pada piksel atas, bawah dan sampingnya [16]. Berikut kernel gaussian blur filter yang digunakan dalam penelitian ini:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{3}$$

2.4.3 Sobel

Filter Sobel termasuk *high-pass* filter yang umum digunakan untuk deteksi tepi pada citra. Sobel memiliki dua kernel untuk deteksi tepi yaitu kernel sobel vertikal untuk mendeteksi tepi secara vertikal dan kernel sobel horizontal yang mendeteksi tepi secara horizontal [8]. Berikut kernel Sobel vertikal dan horizontal yang digunakan dalam penelitian ini:

$$\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} \qquad \begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \tag{4}$$

2.4.4 Laplacian

Filter ini dapat digunakan untuk deteksi tepi pada citra karena sifatnya yang sensitif dengan perubahan intensitas yang cepat [6]. Tidak seperti Sobel yang menggunakan dua kernel untuk mendeteksi tepi secara vertikal dan horizontal, disini hanya digunakan sebuah kernel yang dapat digunakan untuk deteksi tepi secara vertikal dan horizontal sekaligus. Berikut kernel Laplacian yang digunakan dalam penelitian ini:

$$\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix} \tag{5}$$

2.4.5 Sharpening

Sharpening filter digunakan untuk memperjelas detail halus dalam citra atau untuk meningkatkan detail pada citra yang *blur*, baik karena kesalahan ataupun karena efek dari metode akuisisi citra tertentu [18]. Berikut kernel untuk filter *sharpening* yang digunakan dalam penelitian ini:

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix} \tag{6}$$

2.5. Konvolusi

Konsep filter spasial linear mirip seperti konsep konvolusi pada domain frekuensi, dengan alasan tersebut filter spasial linear biasa disebut juga konvolusi sebuah kernel dengan citra digital [5]. Konvolusi pada fungsi *f(x)* dan *g(x)* didefinisikan pada persamaan 7.

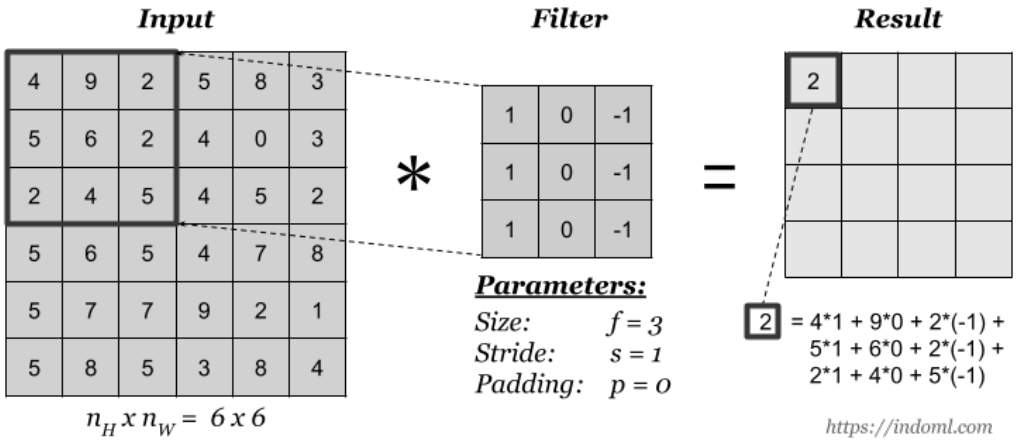
$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da \tag{7}$$

dimana tanda * menyatakan operator konvolusi, dan peubah a adalah peubah bantu. Untuk fungsi diskrit, konvolusi didefinisikan pada persamaan 8.

$$h(x) = f(x) * g(x) = \sum_{a=-\infty}^{\infty} f(a)g(x-a) \tag{8}$$

Pada operasi konvolusi diatas, *g(x)* disebut kernel konvolusi atau filter kernel. Kernel *g(x)* dioperasikan secara bergeser pada sinyal masukan *f(x)*. Jumlah perkalian kedua fungsi pada setiap titik merupakan hasil konvolusi yang dinyatakan dengan keluaran *h(x)* [11].

Pada gambar (2) diilustrasikan bagaimana proses konvolusi pada citra digital yang direpresentasikan dalam bentuk matriks. Operasi konvolusi dilakukan pada matriks input berukuran 6x6 dengan filter berukuran 3x3. Hasil konvolusinya ditampilkan pada matriks *result*.



Gambar 2: *Ilustrasi konvolusi pada citra. Sumber: https://indoml.com*

Jika hasil konvolusi menghasilkan nilai piksel negatif, maka nilai tersebut dijadikan 0, sebaliknya jika hasil konvolusi menghasilkan nilai piksel yang melebihi nilai keabuan maksimum, maka nilai tersebut dijadikan ke nilai keabuan maksimum pada citra tersebut [15].

2.6. Video Stream

Video *stream* dapat dipandang sebagai serangkaian citra digital berturut-turut [19]. Berbeda dengan format video lainya, video *stream* ini tidak disimpan pada media penyimpanan sebagai file dengan format video melainkan langsung disalurkan setiap framenya dari sumber (*source*) ke penerima, dalam hal ini FPGA. Dengan menganggap Video *stream* adalah kumpulan citra digital (*frame*) maka dapat dilakukan metode pengolahan seperti pada citra digital, termasuk penerapan filter spasial.

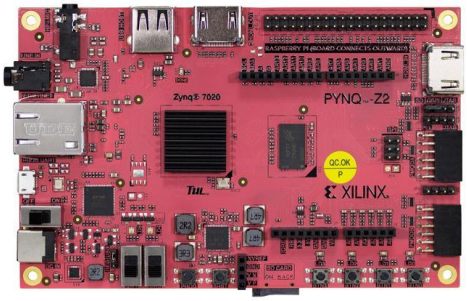
2.7. FPGA

Field Programmable Gate Arrays atau FPGA adalah perangkat semikonduktor yang berbasis *matriks configurable logic block* (CLBs) yang terhubung melalui interkoneksi yang dapat diprogram. FPGA dapat diprogram ulang ke aplikasi atau fungsi yang diinginkan setelah *manufacturing*. Fitur ini yang membedakan FPGA dengan *Application Specific Integrated Circuits* (ASICs), yang dibuat khusus untuk tugas tertentu saja [17].

Sebuah *microprocessor* menerima instruksi berupa kode 1 atau 0, kode-kode ini selanjutnya diinterpretasikan oleh komputer untuk menjalankan perintah yang diberikan. *Microprocessor* ini membutuhkan intruksi berupa kode secara terus menerus untuk menjalankan fungsinya. Sedangkan pada FPGA hanya dibutuhkan sekali konfigurasi *chip* setiap kali dinyalakan. Membuat atau mengunduh *bitstream* yang menentukan fungsi logika dilakukan oleh *logic elements* (LEs), sebuah sirkuit dapat dibuat dengan mengabungkan beberapa LEs menjadi satu kesatuan. Setelah *bitstream* dipasang, FPGA tidak perlu lagi membaca instruksi berupa 1 dan 0, berbeda dengan *microprocessor* yang selalu membutuhkan instruksi [3]. Secara tradisional, untuk membuat sebuah desain FPGA, aplikasi dideskripsikan menggunakan *Hardware Description Language* (HDL) seperti Verilog atau VHDL sehingga menghasilkan sebuah *bitstream* FPGA.

2.7.1 FPGA Development Board

Pada FPGA terdahulu tidak terdapat *processor* (CPU) untuk menjalankan software apapun, sehingga ketika ingin mengimplementasikan aplikasi haruslah merancang sirkuit dari awal, seperti mengonfigurasi FPGA sesederhana gerbang logika OR atau serumit *multi-core processor* [2]. Dewasa ini telah dikembangkan FPGA *Development Board* atau biasa disebut juga FPGA *Board* yaitu teknologi FPGA yang dirangkai dalam sebuah *board* dan dilengkapi dengan *microprocessor* dan beberapa *interface IO* untuk menjalankan tugas tertenu. Umumnya FPGA *Board* telah dilengkapi dengan interface untuk mengakses dan menerapkan desain sirkuitnya. Xilinx, Altera dan Intel adalah produsen FPGA *Board* yang terkenal. FPGA *Board* yang digunakan dalam penelitian ini yaitu Xilinx PYNQ-Z2 dengan Jupyter Notebook sebagai *interface* untuk mengakses dan menjalankan program pada penelitian ini. Bentuk FPGA *Board* Xilinx PYNQ-Z2 dapat dilihat pada gambar (3)



Gambar 3: *FPGA Board Xilinx PYNQ-Z2.*

2.8. Evaluasi Kinerja

Pada penelitian ini peneliti menggunakan waktu komputasi, *frame rate* (FPS), penggunaan CPU, penggunaan memory, penggunaan resident memory (RES), shared memory (SHR), dan virtual memory (VIRT) untuk mengukur kinerja pada penerapan filter spasial linear dengan prosesor ARM dan FPGA.

2.8.1 Waktu Komputasi

Waktu komputasi yang dimaksud oleh peneliti adalah durasi yang dibutuhkan sebuah kernel untuk melakukan filter spasial linear terhadap beberapa *frame* input. Waktu komputasi ini diperoleh dengan cara menghitung selisih waktu selesai dengan waktu dimulai penerapan filter spasial linear pada *frame* input.

$$waktu\ kompuatsi = waktu\ selesai - waktu\ mulai \tag{9}$$

2.8.2 Frame Rate (FPS)

Frame rate atau *frame per second* (fps) adalah banyaknya *frame* yang ditampilkan per detik pada video ataupun video *straem*. Semakin tinggi fps sebuah video maka semakin halus pula gerakan yang dihasilkan. Sebaliknya video dengan fps rendah akan menghasilkan gerakan yang kurang baik. *Frame rate* atau fps dapat dihitung dengan cara membagi jumlah *frame* dengan waktu komputasinya seperti pada persamaan 10 [9].

$$fps = \frac{jumlah\ frame}{waktu\ komputasi} \tag{10}$$

2.8.3 Penggunaan CPU

Pada penelitian ini peneliti menggunakan fitur yang tersedia pada sistem operasi Linux yang berjalan di FPGA Development Board untuk melihat persentase penggunaan CPU pada proses penerapan filter spasial linear. Tampilan dari program ini dapat dilihat pada gambar 4. Program ini menampilkan informasi tentang proses-proses yang berjalan pada sistem operasi seperti ID sebuah proses, user yang menjalankan proses tersebut, *memory* yang digunakan, status sebuah proses, persentase CPU yang digunakan dan lainnya. Status sebuah proses pada program ini dinyatakan dengan beberapa singkatan, diantaranya yaitu:

I = *idle*
R = *running*
S = *sleeping*
Z = *zombie*
D = *uninterruptible sleep*
T = *stopped by job control signal*
t = *stopped by debugger during trace*

Pada CPU yang multi-core sebuah proses akan ditampilkan persentase penggunaan CPUnya berdasarkan core yang digunakan oleh proses tersebut. Jika mode Irix pada program *top* dimatikan, maka program akan berjalan pada mode Solaris di mana penggunaan CPU sebuah proses yang ditampilkan akan dibagi dengan jumlah total core yang ada pada CPU [7].

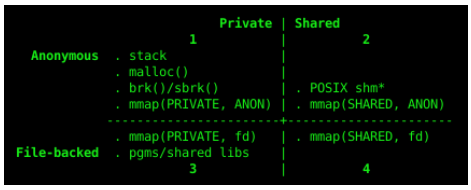
2.8.4 Penggunaan Memory

Pada sistem operasi linux *memory* dibagi menjadi tiga jenis [7]. Pertama yaitu *memory* fisik, sumber daya terbatas di mana kode dan data harus berada saat dijalankan atau direferensikan. Berikutnya adalah *memory swap*, yaitu *memory* yang berguna untuk membantu kerja *memory* fisik, data dari *memory* fisik akan disimpan pada *swap* dan kemudian diambil kembali jika terlalu banyak permintaan pada *memory* fisik. Ketiga yaitu virtual *memory*, sumber daya yang hampir tidak terbatas yang digunakan untuk tujuan berikut [13]:

- *abstraction*, bebas dari alamat / batas *memory* fisik
- *isolation*, setiap proses dalam ruang alamat terpisah
- *sharing*, pemetaan tunggal dapat memenuhi banyak kebutuhan
- *flexibility*, menetapkan alamat virtual ke data

Terlepas dari bentuk *memory* mana yang mungkin digunakan, semua dikelola sebagai *pages* (biasanya 4096 byte). Penggunaan *memory* berhubungan dengan *memory* fisik dan *swap* untuk sistem secara keseluruhan. Untuk setiap proses yang berjalan, setiap *memory* page dibatasi ke satu kuadran seperti pada gambar 5. Baik *memory* fisik dan *memory* virtual dapat menyertakan salah satu dari empat kuadran, sementara file *swap* hanya mencakup kuadran 1 sampai 3. Memori di kuadran 4,

Gambar 4: *Tampilan program top pada Linux.*



Gambar 5: *Kuadran pembagian memory pada Linux.*

bertindak sebagai file *swap* khusus ketika dimodifikasi [7].

2.8.5 Virtual Memory (VIRT)

Virtual *memory* menggunakan disk sebagai perpanjangan dari RAM sehingga ukuran efektif *memory* yang dapat digunakan bertambah secara bersamaan. Kernel akan menulis konten dari blok *memory* yang saat ini tidak digunakan ke hard disk sehingga *memory* dapat digunakan untuk tujuan lain. Ketika konten asli dibutuhkan lagi, mereka dibaca kembali ke dalam *memory*. Ini semua dibuat transparan sepenuhnya bagi pengguna. Program yang berjalan di Linux hanya melihat jumlah *memory* yang tersedia lebih besar dan tidak memperhatikan bahwa sebagian dari program tersebut berada di disk dari waktu ke waktu. Tentu saja, membaca dan menulis hard disk lebih lambat daripada menggunakan *memory* fisik, sehingga program tidak berjalan secepat itu. Bagian dari hard disk yang digunakan sebagai *memory* virtual disebut ruang swap [12].

Kolom VIRT pada program **top** menunjukkan jumlah total *memory* virtual yang digunakan oleh proses. Ini mencakup semua kode, data dan *shared libraries* ditambah dengan *pages* yang telah ditukar dan *pages* yang telah dipetakan tetapi tidak digunakan [7].

2.8.6 Resident Memory (RES)

Resident memory adalah bagian dari ruang alamat virtual (VIRT) yang mewakili *memory* fisik yang tidak ditukar yang sedang digunakan tugas. Resident *memory* ini juga merupakan penjumlahan dari RSan, RSfd dan Bidang RSsh. Ini dapat mencakup private anonymous *pages*, halaman pribadi yang dipetakan ke file (termasuk *program images* dan *shared libraries*) ditambah shared anonymous *pages*. Semua *memory* tersebut didukung oleh file *swap* yang direpresentasikan secara terpisah pada SWAP. Resident *memory* ini juga dapat menyertakan *pages* yang didukung *shared file-backed* yang apabila dimodifikasi, maka akan bertindak sebagai file *swap* khusus dan karenanya tidak akan pernah memengaruhi SWAP [7].

2.8.7 Shared Memory (SHR)

Shared memory adalah bagian dari resident *memory* (RES) yang dapat digunakan oleh proses lain. Termasuk *anonymous pages* dan shared file-backed *pages*. Ini juga termasuk private *pages* dipetakan ke file yang mewakili *program images* dan *shared libraries* [7].

3. Implementasi pada FPGA Development Board

Pada penelitian ini digunakan 6 kernel berbeda berukuran 3x3 untuk penerapan filter spasial linear pada video *stream* 720p 60 FPS. Penerapan ini dilakukan pada FPGA Development Board dengan menggunakan prosesor ARM dan FPGA. Kemudian dilakukan analisis kinerja dari keduanya untuk menunjukkan masing-masing waktu komputasi, FPS, persentase penggunaan CPU, penggunaan *memory*, *resident memory* (RES), *shared memory* (SHR), dan *virtual memory* (VIRT) pada masing-masing kernel.

FPGA Development Board dirangkai seperti yang telah disebutkan pada Bab sebelumnya, pada gambar **??**. HDMI Output pada FPGA dihubungkan ke monitor dan HDMI Input dihubungkan ke *source* dalam hal ini laptop Lenovo Ideapad 320. FPGA Development Board juga dihubungkan ke *router* menggunakan kabel UART agar dapat diakses menggunakan protokol *ssh*. Selanjutnya memasang semua *library* yang dibutuhkan untuk melakukan penerapan filter spasial linear pada video *stream* menggunakan FPGA Development Board.

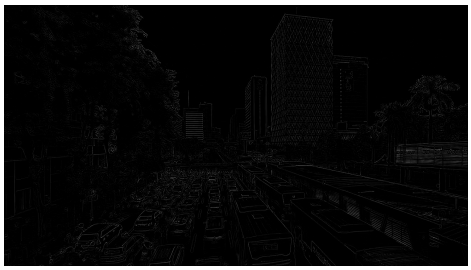
3.1. Penerapan Filter Spasial

Setiap *frame* dari *source* video *stream* dibaca sebagai citra digital yang direpresentasikan sebagai matriks berukuran 1280x720 dengan rentang nilai keabuan pada

Gambar 6: *Contoh Frame Grayscale.*

Gambar 8: *Hasil filter Gaussian Blur.*

Gambar 7: *Hasil filter Average Blur.*



Gambar 9: *Hasil filter Laplacian.*

masing-masing piksel yaitu dari 0 sampai 255. Setiap piksel pada matriks tersebut hanya memiliki satu lapis warna sebab citra yang diterima dari *source* adalah citra *grayscale*, tidak seperti citra warna yang memiliki tiga lapis warna pada setiap pikselnya.

Proses filter spasial dilakukan dengan operasi konvolusi pada setiap matriks dengan kernel yang telah ditentukan sebelumnya. Operasi konvolusi ini menghasilkan matrix baru dengan ukuran 1280x720. Matriks hasil tersebut selanjutnya direpresentasikan kembali sebagai citra digital yang selanjutnya disebut sebagai hasil filter. Hasil filter dari setiap *frame* ini ditampilkan ke monitor melalui HDMI Output pada FPGA Development Board secara berkesinambungan sehingga tampak seperti video.

Salah satu contoh *frame* dari *source* dapat dilihat pada gambar 6. Selanjutnya dilakukan filter spasial menggunakan 6 kernel yang telah ditentukan sebelumnya.

3.1.1 Average Blur

Penerapan filter spasial pada *frame grayscale* yang berukuran 1280x720 pixel dengan kernel *average blur* (2) yang berukuran 3x3 menghasilkan citra *blur* yang berukuran 1280x720. Hasil filter *average blur* dapat dilihat pada gambar 7. Filter seperti ini dapat digunakan untuk mengurangi derau pada citra.

3.1.2 Gaussian Blur

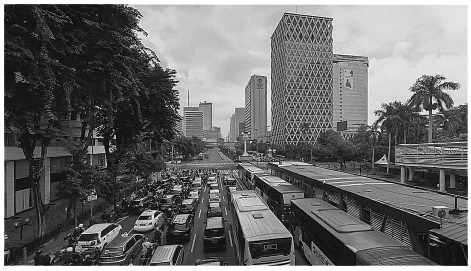
Penerapan filter spasial dengan kernel *gaussian blur* (3) yang berukuran 3x3 menghasilkan citra *blur* yang secara kasat mata mirip dengan filter *average blur*. Namun apabila diperhatikan nilai masing-masing pixel pada gambar 8 akan terlihat berbeda dengan nilai masing-masing pixel pada gambar 7. Hal ini disebabkan oleh nilai bobot pada kernel *gaussian blur* yang berbeda dengan kernel *average blur* sehingga hasil konvolusinya juga berbeda.

3.1.3 Laplacian

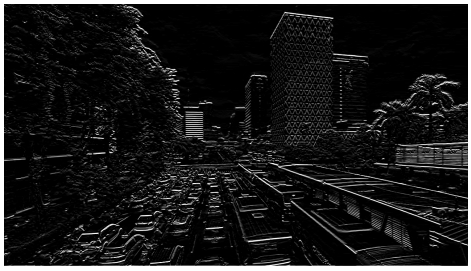
Penerapan filter spasial dengan kernel *laplacian* (5) menghasilkan citra *biner* yang hanya direpresentasikan dengan warna hitam dan putih saja, dapat dilihat pada gambar 9. Filter seperti ini dapat digunakan pada metode deteksi tepi dalam proses pengolahan citra digital.

3.1.4 Sharpening

Penerapan filter spasial dengan kernel *sharpening* (6) dapat meningkatkan detail (seperti garis) pada citra, namun dapat juga dapat menimbulkan derau pada citra apabila bobot kernelnya tidak sesuai. Filter seperti ini lebih tepat digunakan untuk memperbaiki kualitas citra (dengan nilai kernel yang sesuai). Hasil filter *sharpening* ini dapat dilihat pada gambar 10.



Gambar 10: *Hasil filter Sharpening.*



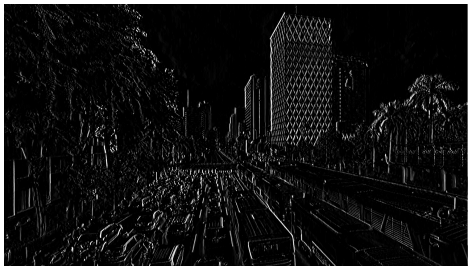
Gambar 11: *Hasil filter Sobel Horizontal.*

3.1.5 Sobel Horizontal

Penerapan filter spasial dengan kernel *sobel horizontal* (4) menghasilkan citra *biner*, dapat dilihat pada gambar 11. Filter seperti lebih tepat digunakan pada metode deteksi tepi dengan citra yang banyak mengandung garis horizontal.

3.1.6 Sobel Vertical

Penerapan filter spasial dengan kernel *sobel vertical* (4) menghasilkan citra *biner*, dapat dilihat pada gambar 12. Sama halnya dengan filter *sobel horizontal*, filter *sobel vertical* juga dapat digunakan untuk metode deteksi tepi, terutama pada citra yang banyak mengandung garis vertikal.



Gambar 12: *Hasil filter Sobel Vertical.*

3.2. Penerapan Filter Spasial dengan Prosesor ARM dan FPGA

Penerapan filter spasial pada prosesor ARM dilakukan dengan menggunakan *library* OpenCV dengan bahasa pemrograman Python yang dijalankan pada FPGA Development Board. Sedangkan untuk penerapan filter spasial pada FPGA dilakukan dengan menggunakan *library* xfOpenCV dari Xilinx. *Library* xfOpenCV ini adalah *library* khusus yang dimodifikasi dari OpenCV sehingga proses komputasinya dapat dilakukan dengan FPGA, bukan dengan prosesor ARM yang pada FPGA Development Board ini.

3.3. Proses Evaluasi Kinerja

Pada penelitian ini proses dalam evaluasi kinerja dilakukan dalam dua tahap. Proses pertama untuk menghitung waktu komputasi dan FPS dari masing-masing kernel dengan prosesor ARM dan FPGA. Kemudian proses kedua untuk mencatat persentase penggunaan CPU dan penggunaan *memory* termasuk *resident memory*, *shared memory* dan *virtual memory*.

3.3.1 Menghitung Waktu Komputasi dan FPS

Pada proses ini peneliti melakukan percobaan dengan menggunakan 50 *frame* dan 200 *frame* dari *source* untuk dihitung FPS dan waktu komputasinya. Percobaan dilakukan sebanyak 5 kali menggunakan 50 *frame* dengan masing-masing kernel pada prosesor ARM dan FPGA, dan 5 kali percobaan menggunakan 200 *frame* dengan masing-masing kernel.

Menghitung waktu komputasi dilakukan dengan cara mencatat waktu mulai dan waktu semua *frame* selesai difilter pada setiap percobaan. Waktu komputasi diperoleh dari selisih antara waktu selesai dengan waktu mulai, seperti pada persamaan 9. Dilakukan dengan 50 *frame* dan 200 *frame* pada masing-masing kernel dengan menggunakan prosesor ARM dan FPGA. Proses ini dilakukan dengan menggunakan *library time* pada bahasa pemrograman Python, dapat dilihat pada gambar 13.

Hasil waktu komputasi ini dicatat dan kemudian digunakan untuk menghitung FPS dari masing-masing percobaan. Selanjutnya FPS dari masing-masing percobaan ini dihitung dengan menggunakan persamaan 10.

```
import time

waktu_start = time.time()

# implementasi
...

waktu_stop = time.time()
waktu_komputasi = waktu_stop - waktu_start
```

Gambar 13: *Menghitung waktu komputasi dengan library time di Python.*

```
import os
print(os.getpid())
```

Gambar 14: *Menampilkan PID sebuah proses dengan bahasa pemrograman Python.*

3.3.2 Mencatat Penggunaan Memory dan CPU (Resource)

Pada proses ini peneliti menggunakan fitur yang tersedia pada sistem operasi Linux yang berjalan di FPGA Development Board untuk membantu mencatat penggunaan CPU dan *memory* (*resource*) pada saat proses penerapan filter spasial. Program ini menampilkan data tentang proses yang berjalan seperti ID sebuah proses, persentase *memory* yang digunakan oleh sebuah proses, persentase CPU yang digunakan, berapa lama sebuah proses berjalan, *resident memory*, *shared memory* dan *virtual memory*.

Peneliti melakukan 5 kali percobaan untuk mencatat penggunaan *memory* dan CPU dari masing-masing kernel dengan prosesor ARM dan FPGA. Percobaan ini dilakukan dengan menggunakan *Jupyter Notebook* yang berjalan pada FPGA Development Board yang dapat diakses menggunakan *web browser* dari perangkat lain. Serta digunakan protokol *ssh* untuk mengakses FPGA Development Board dan menjalankan program untuk menampilkan penggunaan *resource* dari proses yang sedang berjalan.

Pertama peneliti menampilkan ID proses dari *Jupyter Notebook* yang digunakan untuk mencatat penggunaan *memory* dan CPU dari penerapan filter menggunakan prosesor ARM dan FPGA. Cara menampilkan ID proses (PID) menggunakan bahasa pemrograman Python dapat dilihat pada gambar 14. ID proses tersebut kemudian digunakan pada program **top** untuk menampilkan *resource* yang digunakan oleh proses tersebut.

Pada gambar 15 ditunjukan cara menggunakan

```
$ top -d 0.1 -p 4382 -b >> arm-laplacian1.tx
```

Gambar 15: *Menjalankan program **top** kemudian menyimpan hasilnya pada file arm-laplacian1.txt.*

program **top** untuk mencatat penggunaan *resource* dari proses dengan ID 4382 kemudian *output*nya disimpan pada file *arm-laplacian1.txt*. Program **top** tersebut dijalankan sesaat sebelum proses penerapan filter dijalankan pada *Jupyter Notebook*. Output dari program **top** setiap 0.1 detik disimpan pada file text yang telah ditentukan. Setelah seluruh frame selesai difilter maka program **top** juga dihentikan.

4. Analisis Kinerja

4.1. Waktu Komputasi

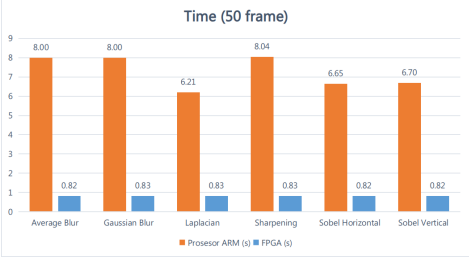
Data waktu komputasi dengan menggunakan 50 frame pada masing-masing kernel dapat dilihat pada tabel 1 dan grafik pada gambar 16. Secara umum waktu komputasi dengan menggunakan prosesor ARM lebih lambat daripada waktu komputasi dengan menggunakan FPGA. Rata-rata waktu komputasi dengan prosesor ARM menggunakan 50 frame adalah 7,26 detik, sedangkan rata-rata waktu komputasi dengan FPGA menggunakan 50 frame hanya 0,82 detik.

Tabel 1: *Tabel perbandingan waktu komputasi dengan menggunakan 50 frame.*

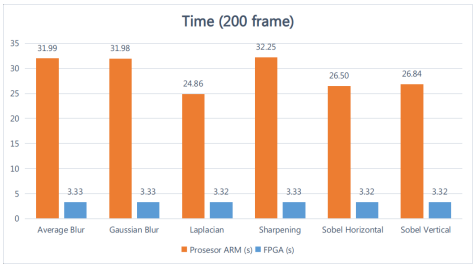
Filter	Prosesor ARM (s)	FPGA (s)
Average Blur	8.003612137	0.823560476
Gaussian Blur	7.998623228	0.827384996
Laplacian	6.210968781	0.827101517
Sharpening	8.041392469	0.825223923
Sobel Horizontal	6.646468353	0.823914003
Sobel Vertical	6.696593809	0.823559713
Rata-rata	7.266276463	0.825124105

Data waktu komputasi menggunakan 200 frame pada maasing-masing kernel dapat dilihat pada tabel 2 dan grafik pada gambar 17. Rata-rata waktu komputasi dengan prosesor ARM menggunakan 200 frame adalah 29,06 detik, sedangkan rata-rata waktu komputasi dengan FPGA menggunakan 200 frame hanya 3,32 detik.

Waktu komputasi tercepat dengan menggunakan



Gambar 16: *Grafik perbandingan waktu komputasi dengan 50 frame dan 200 frame*



Gambar 17: *Grafik perbandingan waktu komputasi dengan 50 frame dan 200 frame*

Tabel 2: *Tabel perbandingan waktu komputasi dengan menggunakan 200 frame.*

Filter	Prosesor ARM (s)	FPGA (s)
Average Blur	31.9899159	3.325525188
Gaussian Blur	31.97958055	3.325351763
Laplacian	24.85993662	3.323753452
Sharpening	32.24906039	3.328786802
Sobel Horizontal	26.49739237	3.32414484
Sobel Vertical	26.84196448	3.324060822
Rata-rata	29.06964172	3.325270478

prosesor ARM terdapat pada filter *laplacian* yaitu 6,21 detik dengan 50 frame dan 24.85 detik dengan 200 frame. Sedangkan waktu komputasi paling lambat ketika menggunakan prosesor ARM terdapat pada filter *sharpening* yaitu 8,04 detik dengan 50 frame dan 32,24 detik dengan 200 frame.

Untuk menghitung efisiensi waktu komputasi yang dimiliki FPGA dibandingkan dengan prosesor ARM, digunakan rumus:

$$= 100\% - \left(\frac{waktu\ komputasi\ FPGA}{waktu\ komputasi\ ARM Prosesor} \times 100\% \right)$$
$$= 100\% - \left(\frac{3,32}{29,06} \times 100\% \right)$$
$$= 100\% - 11.42\%$$
$$= 88.58\%$$

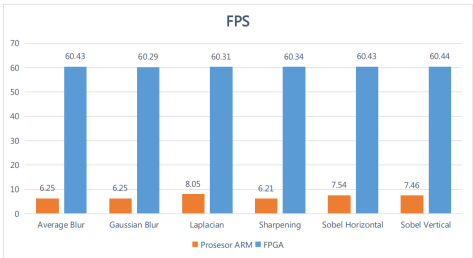
sehingga diperoleh efisiensi waktu komputasi FPGA dibandingkan dengan prosesor ARM adalah sebesar 88.85%.

4.2. Frame Rate (FPS)

Dengan mengetahui waktu komputasi dan jumlah frame maka frame rate atau FPS dapat dihitung menggunakan persamaan 10. Data FPS dari masing-masing kernel dengan prosesor ARM dan FPGA dapat dilihat pada tabel 3 dan grafik pada gambar 18.

Tabel 3: *Tabel perbandingan FPS dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM	FPGA
Average Blur	6.249583533	60.42684593
Gaussian Blur	6.25253493	60.28874396
Laplacian	8.047839131	60.31306253
Sharpening	6.209782985	60.33633034
Sobel Horizontal	7.53538058	60.42633377
Sobel Vertical	7.459019618	60.44047449
Rata-rata	6.959023463	60.37196517



Gambar 18: *Grafik perbandingan FPS dengan menggunakan prosesor ARM dan FPGA.*

Pada tabel 3 terlihat dengan menggunakan prosesor ARM diperoleh rata-rata 6.95 frame per detik (FPS), sedangkan ketika menggunakan FPGA diperoleh rata-rata 60.37 frame per detik. Terlihat pada grafik 18 nilai FPS dengan FPGA jauh lebih tinggi daripada dengan prosesor ARM.

Untuk menghitung efisiensi FPS yang dimiliki FPGA

dibandingkan dengan prosesor ARM, digunakan rumus:

$$
\begin{aligned}
&= 100\% - \left(\frac{FPS\ ARM Prosesor}{FPS\ FPGA} \times 100\% \right) \\
&= 100\% - \left(\frac{6.95}{60.37} \times 100\% \right) \\
&= 100\% - 11.51\% \\
&= 88.49\%
\end{aligned}
$$

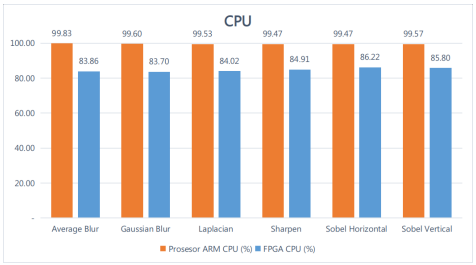
sehingga diperoleh efisiensi FPS dari FPGA dibandingkan dengan prosesor ARM adalah sebesar 88.49%.

4.3. Penggunaan CPU

Data perbandingan penggunaan CPU pada masing-masing kernel dengan prosesor ARM dan FPGA dapat dilihat pada tabel 4 dan grafik pada gambar 19. Rata-rata penggunaan CPU dengan prosesor ARM adalah 99.58% sedangkan dengan FPGA diperoleh 84.75%. Data ini menunjukkan bahwa penggunaan CPU dengan prosesor ARM sedikit lebih besar daripada dengan FPGA.

Tabel 4: *Tabel perbandingan penggunaan CPU dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (%)	FPGA (%)
Average Blur	99.83	83.86
Gaussian Blur	99.60	83.70
Laplacian	99.53	84.02
Sharpening	99.47	84.91
Sobel Horizontal	99.47	86.22
Sobel Vertical	99.57	85.80
Rata-rata	99.58	84.75



Gambar 19: *Grafik perbandingan penggunaan CPU dengan menggunakan prosesor ARM dan FPGA.*

Penggunaan CPU terbesar dengan prosesor ARM yaitu pada kernel *average blur* (99,83%) dan dengan FPGA pada kernel *sobel horizontal* (86,22%). Penggunaan CPU terkecil dengan prosesor ARM yaitu pada kernel *sharpening* dan *sobel horizontal* (99,47%) dan dengan FPGA pada kernel *gaussian blur* (83,70%).

Untuk menghitung efisiensi penggunaan CPU yang dimiliki FPGA dibandingkan dengan prosesor ARM, digunakan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{CPU\ FPGA}{CPU\ ARM Prosesor} \times 100\% \right) \\
&= 100\% - \left(\frac{84.75}{99.58} \times 100\% \right) \\
&= 100\% - 85.11\% \\
&= 14.89\%
\end{aligned}
$$

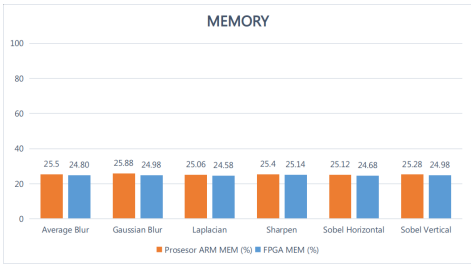
sehingga diperoleh efisiensi penggunaan CPU FPGA dibandingkan dengan prosesor ARM adalah sebesar 14.89%.

4.4. Penggunaan Memory

Data penggunaan *memory* dengan prosesor ARM dan FPGA dapat dilihat pada tabel 5 dan grafik pada gambar 20. Data ini menunjukkan persentase *memory* yang digunakan pada masing-masing kernel. Rata-rata penggunaan *memory* dengan prosesor ARM adalah 25,37% dan 24,86% dengan FPGA.

Tabel 5: *Tabel perbandingan penggunaan memory dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (%)	FPGA (%)
Average Blur	25.50	24.80
Gaussian Blur	25.88	24.98
Laplacian	25.06	24.58
Sharpening	25.40	25.14
Sobel Horizontal	25.12	24.68
Sobel Vertical	25.28	24.98
Rata-rata	25.37	24.86



Gambar 20: *Grafik perbandingan penggunaan memory dengan menggunakan prosesor ARM dan FPGA.*

Walaupun FPGA lebih baik daripada prosesor ARM pada segi waktu komputasi dan FPS namun penggunaan *memory* pada penerapan filter ini terlihat tidak jauh berbeda. Penggunaan *memory* FPGA ini hanya 0,51% lebih rendah dari penggunaan *memory* dengan prosesor ARM.

Efisiensi penggunaan *memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat dihitung dengan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{memory\ FPGA}{memory\ ARM Prosesor} \times 100\% \right) \\
&= 100\% - \left(\frac{24.86}{25.37} \times 100\% \right) \\
&= 100\% - 97.98\% \\
&= 2.02\%
\end{aligned}
$$

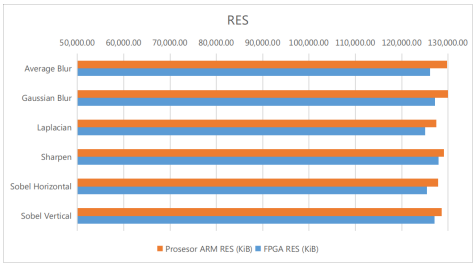
diperoleh efisiensi penggunaan *memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 2.02%.

4.5. Resident Memory (RES)

Data penggunaan *resident memory* atau RES dengan prosesor ARM dan FPGA dapat dilihat pada tabel 6 dan grafik pada gambar 21. Data ini menunjukkan banyaknya RES (dalam satuan *kilobyte*) yang digunakan pada saat penerapan filter spasial pada video *stream* dengan masing-masing kernel.

Tabel 6: *Tabel perbandingan penggunaan resident memory (RES) dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (KiB)	FPGA (KiB)
Average Blur	129794.80	126097.60
Gaussian Blur	131804.40	127135.20
Laplacian	127493.60	125005.60
Sharpening	129117.22	127931.20
Sobel Horizontal	127830.40	125420.00
Sobel Vertical	128611.20	127033.60
Rata-rata	129108.60	126437.20



Gambar 21: *Grafik perbandingan penggunaan resident memory (RES) dengan menggunakan prosesor ARM dan FPGA.*

Rata-rata RES yang digunakan pada prosesor ARM adalah 129108,60 KiB dan 126437,20 KiB pada FPGA. Terlihat bahwa penggunaan RES pada prosesor ARM dan FPGA juga tidak jauh berbeda. Penggunaan RES terbesar dengan prosesor ARM yaitu pada kernel *gaussian blur* 131804,40 KiB, sedangkan dengan FPGA yaitu pada kernel *sharpening* 127931,20 KiB.

Efisiensi penggunaan *resident memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat dihitung dengan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{resident\ memory\ FPGA}{resident\ memory\ ARM Prosesor} \times 100\% \right) \\
&= 100\% - \left(\frac{126437.20}{129108.60} \times 100\% \right) \\
&= 100\% - 97.93\% \\
&= 2.07\%
\end{aligned}
$$

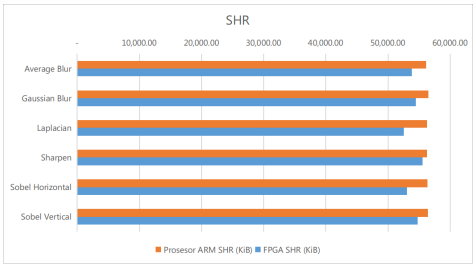
diperoleh efisiensi penggunaan *resident memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 2.07%.

4.6. Shared Memory (SHR)

Data penggunaan *shared memory* dengan prosesor ARM dan FPGA dapat dilihat pada tabel 7 dan grafik pada gambar 22. Data ini menunjukkan banyaknya *shared memory* (dalam satuan *kilobyte*) yang digunakan pada saat penerapan filter spasial pada video *stream* dengan masing-masing kernel. Rata-rata penggunaan

Tabel 7: *Tabel perbandingan penggunaan shared memory (SHR) dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (KiB)	FPGA (KiB)
Average Blur	56,157.40	53,840.00
Gaussian Blur	56,503.60	54,504.80
Laplacian	56,248.00	52,568.00
Sharpen	56,298.82	55,528.80
Sobel Horizontal	56,349.60	53,015.20
Sobel Vertical	56,396.00	54,728.00
Rata-rata	56,325.57	54,030.80



Gambar 22: *Grafik perbandingan penggunaan shared memory (SHR) dengan menggunakan prosesor ARM dan FPGA.*

shared memory pada prosesor ARM adalah 56325,57 KiB dan 54030,80 KiB pada FPGA. Terlihat bahwa penggunaan *shared memory* pada prosesor ARM sedikit lebih besar daripada FPGA. Penggunaan *shared memory* terbesar pada prosesor ARM yaitu pada kernel *gaussian blur* 56503,60 KiB dan kernel *laplacian* 55528,80 KiB pada FPGA. Penggunaan *shared memory* terkecil pada prosesor ARM yaitu pada kernel *average blur* 56157,40 KiB dan kernel *laplacian* 42568 KiB pada FPGA.

Efisiensi penggunaan *shared memory* yang dimiliki FPGA dibandingkan dengan prosesor ARM, dapat dihitung dengan persamaan berikut:

$$= 100\% - \left(\frac{shared\ memory\ FPGA}{shared\ memory\ ARM Prosesor} \times 100\% \right)$$
$$= 100\% - \left(\frac{54030.80}{56325.57} \times 100\% \right)$$
$$= 100\% - 95.92\%$$
$$= 4.08\%$$

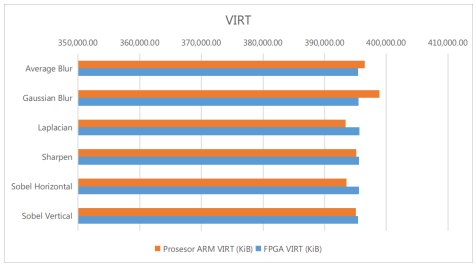
diperoleh efisiensi penggunaan *shared memory* FPGA dibandingkan dengan prosesor ARM adalah sebesar 4.08%.

4.7. Virtual Memory (VIRT)

Data penggunaan *virtual memory* atau VIRT dapat dilihat pada tabel 8 dan grafik pada gambar 23. Data ini menunjukkan banyaknya *virtual memory* (dalam satuan *kilobyte*) yang digunakan pada saat penerapan filter spasial pada video *stream* dengan masing-masing kernel.

Tabel 8: *Tabel perbandingan penggunaan virtual memory (VIRT) dengan menggunakan prosesor ARM dan FPGA.*

Filter	Prosesor ARM (KiB)	FPGA (KiB)
Average Blur	396,474.64	395,396.00
Gaussian Blur	398,862.80	395,488.80
Laplacian	393,388.00	395,638.40
Sharpen	395,126.77	395,575.20
Sobel Horizontal	393,544.80	395,524.00
Sobel Vertical	395,048.80	395,385.60
Rata-rata	395,407.64	395,501.33



Gambar 23: *Grafik perbandingan penggunaan virtual memory (VIRT) dengan menggunakan prosesor ARM dan FPGA.*

Rata-rata penggunaan VIRT pada prosesor ARM adalah 395407,64 KiB dan 395501,33 KiB pada FPGA. Rata-rata penggunaan VIRT pada FPGA sedikit lebih

tinggi dari pada prosesor ARM. Penggunaan VIRT terbesar dengan prosesor ARM yaitu pada kernel *gaussian blur* 398862,80 KiB dan kernel *laplacian* 395638,40 KiB pada FPGA. Penggunaan VIRT terkecil dengan prosesor ARM yaitu pada kernel *laplacian* 393388,00 KiB dan kernel *sobel vertical* 395385,60 KiB pada FPGA.

Efisiensi penggunaan *virtual memory* yang dimiliki prosesor ARM dibandingkan dengan FPGA, dapat dihitung dengan persamaan berikut:

$$
\begin{aligned}
&= 100\% - \left(\frac{virtual\ memory\ ARMProsessor}{virtual\ memory\ FPGA} \times 100\% \right) \\
&= 100\% - \left(\frac{395407.64}{395501.33} \times 100\% \right) \\
&= 100\% - 99.97\% \\
&= 0.03\%
\end{aligned}
$$

Ini menunjukkan bahwa penggunaan *virtual momory* pada prosesor ARM hanya 0.03% lebih baik dari penggunaan *virtual momory* pada FPGA.

5. Kesimpulan

Berdasarkan hasil penerapan filter spasial linear pada FPGA Development Board dengan menggunakan 6 kernel, peneliti dapat menarik beberapa kesimpulan sebagai berikut:

1. Proses implementasi filter spasial linear pada video *stream* dengan FPGA Development Board dilakukan dengan *library* OpenCV *python* dan *library* xfOpenCV Xilinx. Setiap *frame* dari *source* video *stream* direpresentasikan sebagai citra digital kemudian dilakukan filter spasial linear, selanjutnya hasil filter ini ditampilkan secara berkesinambungan sehingga tampak seperti video.
2. Waktu komputasi dan FPS dengan menggunakan FPGA secara umum lebih baik dibandingkan dengan menggunakan prosesor ARM. Penggunaan CPU pada FPGA sedikit lebih rendah dibandingkan penggunaan CPU pada prosesor ARM. Secara umum penggunaan *memory*, *shared memory*, *virtual memory*, dan *resident memory* pada FPGA tidak jauh berbeda dengan yang digunakan pada prosesor ARM.

DAFTAR PUSTAKA

[1] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. "Performance comparison of FPGA, GPU and CPU in image processing". In: *2009 International Conference on Field Programmable Logic and Applications*. 2009, pp. 126–131. DOI: 10.1109/FPL.2009.5272532.

[2] Priyabrata Biswas. "Introduction to FPGA and its Architecture". https://towardsdatascience.com/introduction-to-fpga-and-its-architecture-20a62c14421c. Accessed on 2020-06-18. 2019.

[3] Peter Cheung. "Introduction to FPGAs". http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/Lecture2-IntroductiontoFPGAs.pdf. Accessed on 2020-04-19. 2019.

[4] Jason Cong dkk. "Understanding Performance Differences of FPGAs and GPUs". In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: Association for Computing Machinery, 2018. ISBN: 9781450356145. DOI: 10.1145/3174243.3174970.

[5] Rafael C. Gonzalez and Richard E. Woods. "Digital Image Processing". 2nd. ISBN-13: 978-0201180756. Upper Saddle River, New Jersey 07458: Prentice Hall, 2001.

[6] Xu Jingbo dkk. "A New Method for Realizing LOG Filter in Image Edge Detection". In: *The 6th International Forum on Strategic Technology* (Aug. 2011). DOI: 10.1109/IFOST.2011.6021127.

[7] Michael Kerrisk. "(Top) Linux Manual Page". https://www.man7.org/linux/man-pages/man1/top.1.html. Accessed on 2021-02-2. 2020.

[8] Marcin Kowalczyk, Dominika Przewlocka, and Tomasz Krvjak. "Real-Time Implementation of Contextual Image Processing Operations for 4K Video Stream in Zynq UltraScale+ MPSoC". In: *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)* (Oct. 2018). DOI: 10.1109/DASIP.2018.8597105.

[9] Pavan C. Madhusudana dkk. "Capturing Video Frame Rate Variations via Entropic Differencing". In: *IEEE Signal Processing Letters* 27 (2020), pp. 1809–1813. DOI: 10.1109/LSP.2020.3028687.

[10] Darma Putra. "Pengolahan Citra Digital". ISBN-13: 978-979-29-1443-6. Jl. Beo 38-40, Yogyakarta 55281: Penerbit Andi, 2010.

[11] Munir Rinaldi. “Pengolahan Citra Digital dengan Pendekatan Algoritmik”. ISBN: 979-3338296. Bandung: Penerbit Informatika, 2004.

[12] Lars S dkk. “The Linux System Adminstrator’s Guide Chapter 6. Memory Management”. https://tldp.org/LDP/sag/html/vm-intro.html. Accessed on 2021-02-22. 2020.

[13] Avi Silbershatz, Peter Baer Galvin, and Greg Gagne. “Operationg System Concepts”. ISBN: 978-0-470-12872-5. John Wiley and Sons, Inc, 2009.

[14] Eduardo A.B. da Silva and Gelson V. Mendonca. “4 - Digital Image Processing”. In: *The Electrical Engineering Handbook*. Ed. by Wai-Kai Chen. Burlington: Academic Press, 2005, pp. 891–910. ISBN: 978-0-12-170960-0. DOI: https://doi.org/10.1016/B978-012170960-0/50064-5.

[15] T. Sutoyo dkk. “Teori Pengolahan Citra Digital”. ISBN-13: 978-979-29-0974-6. Jl. Beo 38-40, Yogyakarta 55281: Penerbit Andi, 2009.

[16] Dmitry I. Ustyukov, Alex I. Efimov, and Dmitry A. Kolchaev. “Features of Image Spatial Filters Implementation on FPGA”. In: *Mediterranean Conference On Embedded Computing (Meco)* (June 2019).

[17] Xilinx. “Field Programmable Gate Array (FPGA)”. https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. Accessed on 2020-04-17. 2020.

[18] Ching-Chung Yang. “Finest Image Sharpening by Sse of the Modified Mask Filter Dealing with Highest Spatial Frequencies”. In: *OPTIK* (Sept. 2013). DOI: 10.1016/j.ijleo.2013.09.070.

[19] Jin Zhao. “Video/Image Processing on FPGA”. Master thesis. Worcester Polytechnic Institute, Apr. 2015.