UNIVERSITAS INDONESIA

OPENCL ACCELERATOR UNTUK DEEP LEARNING INFERENCE PADA PERANGKAT MOBILE

TUGAS AKHIR

TSESAR RIZQI PRADANA
1406543725

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JANUARI 2018

UNIVERSITAS INDONESIA

OPENCL ACCELERATOR UNTUK DEEP LEARNING INFERENCE PADA PERANGKAT MOBILE

TUGAS AKHIR
Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer

TSESAR RIZQI PRADANA
1406543725

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JANUARI 2018

HALAMAN PERSETUJUAN

Judul	:	OpenCL Accelerator untuk Deep Learning Inference pada Perangkat Mobile
Nama	:	Tsesar Rizqi Pradana
NPM	:	1406543725

Laporan Tugas Akhir ini telah diperiksa dan disetujui.

20 Januari 2018

Prof. T. Basaruddin

Pembimbing Tugas Akhir

HALAMAN PERNYATAAN ORISINALITAS

Tugas Akhir ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Tsesar Rizqi Pradana**
NPM	:	**1406543725**
Tanda Tangan	:	
Tanggal	:	**20 Januari 2018**

HALAMAN PENGESAHAN

Tugas Akhir ini diajukan oleh	:	
Nama	:	Tsesar Rizqi Pradana
NPM	:	1406543725
Program Studi	:	Ilmu Komputer
Judul Tugas Akhir	:	Optimisasi Inference pada Convolutional Neural Network melalui Operation Graph

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing	:	Prof. T. Basaruddin	( )
Penguji	:		( )
Penguji	:		( )
Penguji	:		( )

Ditetapkan di	:	Depok
Tanggal	:	20 Januari 2018

KATA PENGANTAR

Puji dan syukur Penulis panjatkan kepada Tuhan Yang Maha Esa. Berkat Rahmat-Nya Tugas Akhir yang berjudul ”OpenCL Accelerator untuk Deep Learning Inference pada Perangkat Mobile” ini dapat diselesaikan.

Banyak kendala yang dialami Penulis dalam menyelesaikan Tugas Akhir ini. Namun, Penulis dapat mengatasinya berkat bantuan dari dosen pembimbing, orang tua, teman-teman, dan pihak-pihak lainnya.

Penelitian ini disusun dalam waktu yang cukup singkat sehingga masih terdapat banyak kekurangan baik itu pada konten penelitian maupun pada penulisan. Penulis mengharapkan kritik dan saran dari pembaca sebagai bahan pembelajaran bagi Penulis untuk menyusun karya-karya di kemudian hari.

Melalui kata pengantar ini Penulis juga ingin mengucapkan banyak terimakasih kepada pihak-pihak yang telah membantu penyelesaian Tugas Akhir ini, antara lain:

1. Orang tua yang telah memberikan dukungan moral dan materiil,
2. Prof. T. Basaruddin sebagai Pembimbing I,
3. Bapak Risman Adnan sebagai Pembimbing II,
4. Ryorda Triaptahadi sebagai rekan penelitian, dan
5. Teman-teman semua yang telah memberikan dukungan moral.

Semoga pihak-pihak yang telah disebutkan mendapatkan balasan dari Tuhan Yang Maha Esa atas bantuan mereka dalam penyusunan Tugas Akhir ini. Penulis berharap Tugas Akhir ini dapat memberikan manfaat yang positif kepada berbagai pihak.

Depok, 20 Desember 2017

Tsesar Rizqi Pradana

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Tsesar Rizqi Pradana
NPM	:	1406543725
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Tugas Akhir

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

OpenCL Accelerator untuk Deep Learning Inference pada Perangkat Mobile

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 20 Januari 2018

Yang menyatakan

(Tsesar Rizqi Pradana)

ABSTRAK

Nama	:	Tsesar Rizqi Pradana
Program Studi	:	Ilmu Komputer
Judul	:	OpenCL Accelerator untuk Deep Learning Inference pada Perangkat Mobile

Deep Learning has been widely used in many Artificial Intelligence (AI) application because of its high accuracy. With its impressive advances in AI fields, many developers have started to create innovations on Deep Learning such as creating the possibility to to run Deep Learning on mobile device. Because of its high computational cost and the ability of today's mobile device, it is only possible to do the inference step of Deep Learning. Even for inference, current mobile Deep Learning libraries are not optimal yet because of the lack of support from developers. Popular libraries like Tensorflow Mobile or Tensorflow Lite only supports Deep Learning inference on mobile CPU. In this paper we try to create an accelerator for mobile Deep Learning inference by implementing OpenCL code to support Deep Learning inference on mobile GPU. We want to evaluate the effects of applying the accelerator on mobile Deep Learning inference performance with expectation that it will produce a good performance. We also want to compare the accelerators performance to the original CPU implementations performance.

We have implemented OpenCL code on some Deep Learning inference operations including matrix multiplication, matrix convolution, matrix transpose, and vector addition. We test our implementation using Tensorflow Lite demo application on Android devices with some different Convolutional Neural Network models including Inception, LeNet, and MobileNet. The result shows that Deep Learning Inference operations on OpenCL can achieve much better running time for large enough matrix. However, our accelerator could perform much slower than the original Tensorflow Lite functions for small matrix. This is caused by the time taken for OpenCL setup operations like memory allocation and load variable values to GPU memory are much longer than the actual inference operations themself. This can be seen from the similarity of running time on each operations for small matrix, which indicate the bottleneck on OpenCL setup operations. Further optimization on the OpenCL kernels has been applied and it still does not omit the bottleneck.

Kata Kunci:

@todo
Tuliskan kata kunci yang berhubungan dengan laporan disini

ABSTRACT

Name	:	Tsesar Rizqi Pradana
Program	:	Computer Science
Title	:	OpenCL Accelerator for Mobile Deep Learning Inference

Deep Learning has been widely used in many Artificial Intelligence (AI) application because of its high accuracy. With its impressive advances in AI fields, many developers have started to create innovations on Deep Learning such as creating the possibility to to run Deep Learning on mobile device. Because of its high computational cost and the ability of today's mobile device, it is only possible to do the inference step of Deep Learning. Even for inference, current mobile Deep Learning libraries are not optimal yet because of the lack of support from developers. Popular libraries like Tensorflow Mobile or Tensorflow Lite only supports Deep Learning inference on mobile CPU. In this paper we try to create an accelerator for mobile Deep Learning inference by implementing OpenCL code to support Deep Learning inference on mobile GPU. We want to evaluate the effects of applying the accelerator on mobile Deep Learning inference performance with expectation that it will produce a good performance. We also want to compare the accelerators performance to the original CPU implementations performance.

We have implemented OpenCL code on some Deep Learning inference operations including matrix multiplication, matrix convolution, matrix transpose, and vector addition. We test our implementation using Tensorflow Lite demo application on Android devices with some different Convolutional Neural Network models including Inception, LeNet, and MobileNet. The result shows that Deep Learning Inference operations on OpenCL can achieve much better running time for large enough matrix. However, our accelerator could perform much slower than the original Tensorflow Lite functions for small matrix. This is caused by the time taken for OpenCL setup operations like memory allocation and load variable values to GPU memory are much longer than the actual inference operations themself. This can be seen from the similarity of running time on each operations for small matrix, which indicate the bottleneck on OpenCL setup operations. Further optimization on the OpenCL kernels has been applied and it still does not omit the bottleneck.

Keywords:

@todo
Write up keywords about your report here.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

Karya tulis yang berjudul ”OpenCL Accelerator untuk Deep Learning Inference pada Perangkat Mobile” ini didahului dengan pembahasan mengenai latarbelakang penelitian, permasalahan yang ingin diselesaikan, tujuan penelitian, serta metodologi dari penelitian.

1.1 Latar Belakang

Deep Learning saat ini telah menjadi trend dalam implementasi aplikasi-aplikasi *Artificial Intelligence* seperti klasifikasi gambar, pengenalan bahasa, dan sebagainya. Hal ini disebabkan karena Deep Learning mampu memberikan akurasi yang tinggi jika dibandingkan metode *Machine Learning* biasa. Deep Learning sebenarnya merupakan cabang dari *Machine Learning* yang memanfaatkan *Neural Network* untuk melakukan prediksi suatu label atau skor [1]. Arsitektur *Neural Network* yang digunakan pada *Deep Learning* (*Deep Neural Network*) memiliki lebih dari satu *hidden layer*. *Deep Learning* terdiri dari dua tahap yaitu *training* (latihan) dan inference. Pada tahap *inference* dilakukan prediksi label atau skor dari suatu data masukan menggunakan model *Neural Network* yang telah dilatih pada tahap *training*.

Dengan meningkatnya popularitas Deep Learning, banyak pengembang perangkat lunak yang mulai berlomba dalam menciptakan inovasi pada bidang Deep Learning. Salah satu inovasi yang menarik adalah penerapan Deep Learning pada perangkat *mobile* sehingga aplikasi-aplikasi Deep Learning dapat diakses di mana saja dan kapan saja dengan praktis. Tesorflow [7] merupakan contoh *Deep Learning framework* yang menyediakan dukungan untuk perangkat *mobile* melalui *library* Tensorflow Mobile dan Tensorflow Lite. Penerapan Deep Learning pada perangkat *mobile* memiliki tantangan tersendiri. Deep Learning dikenal memiliki beban komputasi yang sangat besar karena mengandung sangat banyak operasi-operasi matriks. Sehingga dengan kemampuan perangkat *mobile* yang kita saat ini, Deep Learning hanya mungkin dilakukan untuk tahap *inference* saja. Saat ini masih sangat sedikit dukungan yang diberikan dari pengembang perangkat lunak untuk *Deep Learning inference* pada perangkat mobile. Mayoritas dukungan masih terfokus pada perangkat desktop atau laptop yang memang memiliki fleksi-

bilitas lebih tinggi untuk *Deep Learning* . Hampir semua penerapan *Deep Learning* pada perangkat mobile saat ini hanya memanfaatkan CPU untuk komputasi operasi-operasi *inference* .

Mengetahui hal ini, penulis tertarik untuk mencoba menerapkan pemrograman GPU pada operasi-operasi *Deep Learning inference* . Penulis berhipotesis bahwa GPU dapat membantu meningkatkan performa karena beberapa hal. Pertama, GPU memiliki sangat banyak unit pemrosesan yang yang dapat mengeksekusi suatu pekerjaan secara paralel. Meskipun secara individu unit pemrosesan tersebut lebih lemah dari unit pemrosesan pada CPU, dengan teknik paralelisasi suatu pekerjaan besar dapat dibagi ke banyak unit pemrosesan sehingga secara keseluruhan komputasi dapat berjalan lebih cepat. Kedua, operasi-operasi *Deep Learning inference* merupakan operasi-operasi matriks yang sangat mungkin diterapkan secara paralel. Misalnya pada operasi penjumlahan matriks, masing-masing penjumlahan antara dua elemen dari dua matriks pada baris dal kolom yang sama dapat diproses oleh suatu unit pemrosesan tersendiri secara independen dari elemen-elemen lainnya.

Untuk menerapkan penggunaan GPU pada operasi-operasi *Deep Learning inference* penulis menggunakan OpenCL yang merupakan antarmuka pemrograman paralel dan *multi-device*. OpenCL API menggunakan bahasa pemrograman C atau C++ sehingga dapat diterapkan pada perangkat mobile. Untuk *framework Deep Learning* , penulis memilih Tensorflow beserta Tensorflow Lite dalam penelitian ini karena merupakan *framework Deep Learning* paling populer dan memiliki banyak fitur yang salah satunya memungkinakan penggunaan berbagai macam model *Neural Network* pada proses *inference* . Tensorflow sendiri diimplementasikan dengan bahasa C++ sehingga sesuai dengan OpenCL.

1.2 Permasalahan

Pada bagian ini akan dijelaskan mengenai definisi permasalahan yang Penulis hadapi dan ingin diselesaikan serta asumsi dan batasan yang digunakan dalam menyelesaikannya.

1.2.1 Definisi Permasalahan

Berikut adalah permasalahan-permasalahan yang mendorong dilaksanakannya penelitian ini.

1. Apakah OpenCL dapat diterapkan pada Tensorflow Lite untuk komputasi dengan menggunakan *mobile* GPU?

2. Bagaimana pengaruh penerapan pemrograman GPU untuk *inference* pada model-model *Deep Learning* pada perangkat *mobile* ?

3. Bagaimana perbandingan kecepatan dan penggunaan memori pada operasi-operasi *Deep Learning inference* pada perangkat *mobile* sebelum dan sesudah penerapan OpenCL?

4. Operasi-operasi *Deep Learning inference* apa saja yang lebih baik dijalankan pada *mobile* GPU?

1.2.2 Batasan Permasalahan

Batasan-batasan penelitian pada Tugas Akhir ini antara lain:

1. Implementasi OpenCL hanya berlaku untuk perangkat *Android* saja.

2. OpenCL hanya diterapkan pada beberapa operasi *Deep Learning inference* pada kode sumber Tensorflow Lite antara lain perkalian matriks, konvolusi matriks, penjumlahan matriks, dan transpose matriks.

3. Pengujian hanya dilakukan terhadap beberapa arsitektur *Convolutional Neural Network* antara lain Inception, LeNet, dan MobileNet yang telah mencakup semua operasi-operasi penting di Deep Learning.

4. Penulis mengasumsikan bahwa perangkat yang digunakan hanya menggunakan sumber daya multi-core CPU dan GPU beserta memorinya, terlepas dari dorongan performa yang berasal dari perangkat tambahan.

1.3 Tujuan

Tujuan dari penelitian ini adalah sebagai berikut.

1. Mengimplementasikan OpenCL code untuk penggunaan mobile GPU pada operasi-operasi Neural Network inference.

2. Mengetahui pengaruh penggunaan mobile GPU melalui OpenCL pada proses inference pada model-model Deep Learning.

3. Membandingkan performa dan penggunaan memori pada proses inference sebelum dan sesudah penerapan OpenCL pada perangkat mobile.

4. Mengetahui operasi-operasi Deep Learning inference pada yang lebih baik bila dijalankan pada GPU melalui OpenCL.

1.4 Metodologi Penelitian

Metodologi penelitian pada Tugas Akhir ini adalah sebagai berikut.

1. Menentukan Tensorflow Lite dan OpenCL sebagai *framework* dan API yang digunakan untuk melakukan penelitian.
2. Mempelajari kode sumber Tensorflow Lite.
3. Mempelajari OpenCL beserta teknik pemrograman GPU.
4. Mengimplementasikan OpenCL code untuk beberapa operasi *Deep Learning inference* .
5. Melakukan eksperimen menggunakan hasil implementasi.
6. Mengoptimalkan implementasi sebelumnya.
7. Melakukan eksperimen menggunakan hasil implementasi yang sudah dioptimalkan.
8. Melaporkan hasil eksperimen sebagai karya tulis.

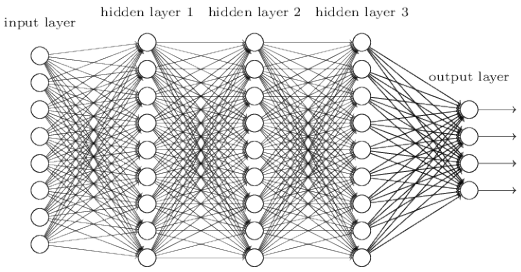
BAB 2
LANDASAN TEORI

Bagian ini menjelaskan teori-teori yang digunakan dalam penelitian. Teori yang dimaksud adalah pengetahuan yang terkait dengan pelaksanaan penelitian.

2.1 Deep Learning

Deep Learning merupakan algoritma *Machine Learning* yang dalam prosesnya memanfaatkan arsitektur *Neural Network* . Cara kerja arsitektur ini terinspirasi oleh struktur otak manusia yang tersusun dari neuron-neuron. Berbeda dengan Artificial Neural Network biasa, arsitektur Neural Network pada Deep Learning memiliki struktur yang lebih kompleks. Ide utama dari Deep Learning adalah bagaimana komputer dapat belajar dari pengalaman dengan cara melatih *Neural Network* menggunakan data-data yang berjumlah besar. *Deep Learning* merupakan salah satu bentuk dari *Supervised Learning* yang berarti *Neural Network* dilatih menggunakan data yang telah diketahui labelnya. *Neural Network* yang telah dilatih dapat digunakan untuk menentukan label atau skor dari data-data baru. Akurasi dari nilai yang dikeluarkan bergantung pada jenis arsitektur dan data-data yang digunakan untuk melatih *Neural Network* tersebut. Dalam Deep Learning terdapat berbagai macam arsitektur yang dapat digunakan pada berbagai macam keperluan prediksi label atau skor. Beberapa jenis arsitektur yang populer adalah Deep Neural Network, Convolutional Neural Network, dan Recurrent Neural Network. Deep Neural Network.

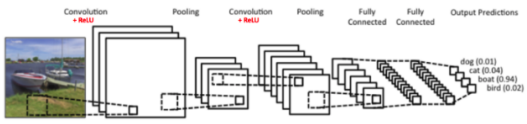
Deep Neural Network (DNN) merupakan jenis arsitektur Deep Learning yang paling sederhana. DNN merupakan *Neural Network* yang terdiri dari satu input layer, satu atau lebih hidden layer, dan satu output layer. Layer-layer pada arsitektur ini merupakan fully connected layer dimana setiap layer tersusun dari node-node yang masing-masing terhubung dengan masing-masing node dari layer-layer tetangganya. Gambar menunjukkan contoh arsitektur DNN.



Gambar 2.1: Contoh Arsitektur Deep Neural Netowork.

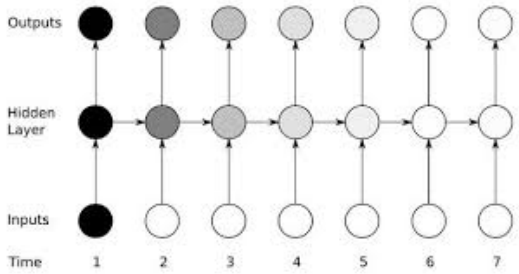
Karena arsitekturnya yang sederhana, DNN pada banyak kasus memiliki performa yang tidak sebaik arsitektur-arsitektur lain. DNN lebih tepat digunakan untuk melakukan tugas-tugas prediksi sederhana yang fitur-fitur dari data yang akan diprediksi sudah diketahui.

Convolutional Neural Network (CNN) merupakan arsitektur *Deep Learning* yang sering digunakan dalam bidang pengolahan citra. CNN pada umumnya terdiri dari beberapa convolution layer, pooling layer, dan fully-connected layer. Convolution layer dan Pooling layer merupakan beberapa jenis layer pada arsitektur ini yang tidak terdapat pada arsitektur-arsitektur lain. Convolution layer biasanya berfungsi untuk mengekstrak fitur-fitur dari data dua dimensi atau tiga dimensi yang masuk. Sedangkan pooling layer berfungsi mereduksi ukuran data output dari convolution layer dengan cara melakukan sampling terhadap data tersebut. Fully-connected layer diletakkan pada akhir arsitektur yang memiliki fungsi untuk melakukan prediksi, sama seperti DNN, dengan menggunakan fitur-fitur hasil ekstraksi convolution layer. Gambar merupakan contoh arsitektur CNN.



Gambar 2.2: Contoh Arsitektur *Convolutional Neural Network* .

Recurrent Neural Network (RNN) merupakan arsitektur Deep Learning yang dapat memproses data yang bersifat sekuensial. Arsitektur ini terdiri dari input layer, hidden layer, dan output layer yang mengandung loop, dimana hasil prediksi dari output layer pada suatu waktu digunakan untuk mengubah parameter dari layer-layer sebelumnya untuk melakukan prediksi pada waktu berikutnya. Dapat dikatakan bahwa output dari suatu layer pada suatu waktu bergantung pada output dari layer itu sendiri pada waktu-waktu sebelumnya. Gambar merupakan contoh arsitektur RNN.



Gambar 2.3: Contoh Arsitektur Recurrent Neural Netowork.

RNN sangat populer dalam bidang Natural Language Processing (NLP) karena kemampuannya mengolah bahasa baik berupa audio atau teks. Hal ini karena bahasa berupa teks atau audio merupakan data yang bersifat sekuensial.

Dengan melihat contoh-contoh arsitektur *Deep Learning* di atas, sekilas dapat diketahui bahwa Deep Learning memiliki beban komputasi yang cukup besar, terutama pada CNN, karena melibatkan banyak parameter dan banyak layer. Oleh karena itu, penggunaan CPU saja dirasa kurang cukup dalam menjalankan operasi-operasi pada Deep Learning.

2.2 Deep Learning Training dan Inference

Sama seperti *Neural Network* biasa, arsitektur *Neural Network* dalam Deep Learning memiliki parameter *weight* pada setiap layer yang digunakan untuk menghitung skor dari input yang masuk ke suatu *layer* dan meneruskan hasilnya ke *layer* berikutnya. Deep Learning terdiri dari dua tahap yaitu training dan inference. Pada tahap training, model *Neural Network* dilatih menggunakan sekumpulan data. Dalam hal ini yang dilatih adalah parameter *weight* hingga menemukan parameter yang optimal.

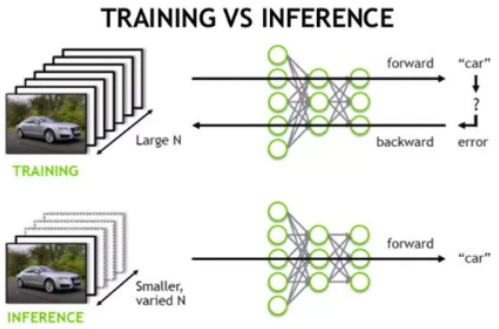
Untuk mengawali proses *training* , *weight* awal diberikan terlebih dahulu kepada model. Biasanya *weight* awal ini merupakan angka-angka acak berdasarkan suatu distribusi tertentu. Kemudian, model dengan *weight* awal yang belum optimal tersebut digunakan untuk melakukan prediksi label atau skor dari data dengan melakukan *forward pass* sepanjang model dari *layer* pertama (input *layer*) hingga mencapai *layer* terakhir (*output layer*).

Hasil prediksi kemudian dibandingkan dengan label aslinya. Nilai *error* dari prediksi dapat dihitung menggunakan fungsi tertentu, misalnya *Mean Square Error*. Informasi *error* ini kemudian dikirim kembali ke *layer -layer* sebelumnya dan akan digunakan oleh *layer -layer* terebut untuk memperbarui parameter *weight* yang belum optimal tadi berdasarkan informasi *error* yang diterima. *Weight* dapat

diperbarui menggunakan algoritma atau fungsi pembaruan *weight* tertentu. Proses prediksi dan mengembalikan *error* ke belakang ini disebut *back propagation*.

Setelah *training* selesai, model *Neural Network* dapat digunakan untuk melakukan proses *inference* . Tahap *inference* merupakan tahap dimana model yang telah dilatih digunakan untuk melakukan prediksi. Proses prediksi ini sama dengan prediksi ketika melakukan *training* yaitu dengan melakukan *forward-pass* terhadap input data mulai dari *input layer* hingga *output layer* pada model.

Letak perbedaan proses *inference* dan *training* pada *Deep Learning* adalah pada tahap *back-propagation*. Setelah melakukan prediksi pada proses *inference* , tidak lagi dilakukan *back-propagation* terhadap *error* hasil prediksi seperti pada proses *training* . Hal ini karena tujuan dari *inference* adalah mendapatkan hasil prediksi skor atau label yang dikeluarkan oleh model yang telah dilatih. Gambar 2.1 menunjukkan perbedaan proses *training* dan *inference* pada *Neural Network* . Terlihat bahwa *training* melakukan pemrosesan data pada dua arah, sedangkan *inference* hanya satu arah.



Gambar 2.4: Perbedaan proses *training* dan *inference* pada *Deep Learning* .

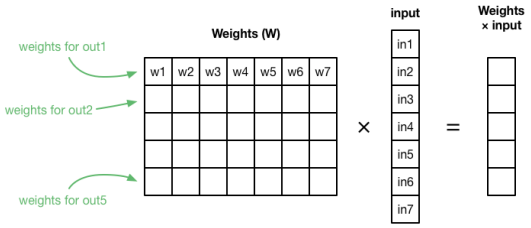
2.3 Operasi-operasi Deep Learning Inference

Tahap inference pada Deep Learning melibatakan banyak sekali operasi matriks yang biasanya merupakan operasi antara input data pada suatu layer dengan weight pada layer itu. Beberapa operasi bahkan memiliki kompleksitas yang tinggi. Berikut adalah beberapa contoh operasi penting yang sering muncul pada Deep Learning inference.

2.3.1 Perkalian Matriks

Operasi perkalian matriks pada Deep Learning inference merupakan salah satu operasi yang memiliki beban komputasi terbesar. Perkalian matriks dapat terjadi

pada bagian fully-connected layer. Pada suatu fully-connected layer ke-l, parameter weight disimpan dalam bentuk matriks yang berukuran MxN dimana M adalah banyaknya node pada layer ke-l dan N adalah banyaknya node pada layer ke-(l-1). Baris ke-i pada weight matriks dari layer ke-l tersebut merupakan weight dari node ke-i pada layer ke-l. Untuk menentukan nilai node-node pada layer ke-l, matrix MxN tersebut dikalikan dengan matrix Nx1 yang berisi nilai-nilai node layer ke-(l-1) sehingga menghasilkan matriks Mx1. Kemudian activation function diterapkan terhadap setiap elemen matriks Mx1 tersebut dan kemudian bias juga ditambahkan jika ada sehingga menghasilkan matriks Mx1 yang merupakan nilai node-node layer ke-i. Gambar adalah ilustrasi dari operasi perkalian matriks pada fully-connected layer.



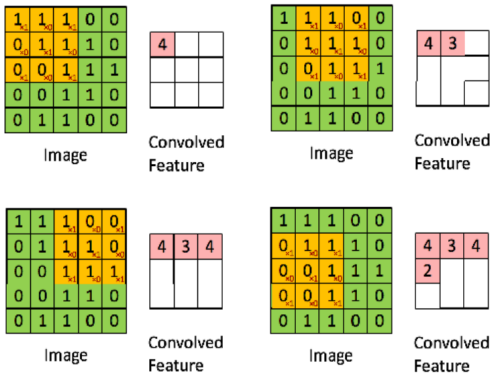
Gambar 2.5: Perkalian matriks pada fully-connected layer saat mengalikan weight dengan input.

2.3.2 Konvolusi Matriks

Operasi konvolusi terjadi pada convolution layer dari CNN. Konvolusi merupakan penerapan filter atau kernel terhadap suatu input dengan mengkonvolusikan kernel tersebut terhadap input sehingga menghasilkan output. Input dan output suatu convolution layer merupakan matriks 3 dimensi yang merepresentasikan lebar, tinggi, dan kedalaman (biasa disebut channel). Sementara filter dinyatakan dalam matriks 4 dimensi yang merepresentasikan lebar filter, tinggi filter, kedalaman filter, dan banyaknya filter. Ukuran tinggi dan lebar filter selalu lebih kecil atau sama dengan tinggi dan lebar intput, sedangkan kedalamannya harus sama dengan kedalaman input.

Pada saat melakukan konvolusi terhadap input, masing-masing elemen pada suatu filter dikalikan dengan masing-masing elemen matriks input yang bersesuaian dengan posisi filter tersebut (seperti dot product pada vector) pada suatu waktu. Hasil dari satu kali operasi tersebut merupakan satu elemen dari matriks output pada posisi yang bersesuaian. Sehingga, setelah proses konvolusi selesai untuk satu filter, akan terbentuk satu lapis output dua dimensi. Jika semua filter sudah diterapkan maka akan terbentuk output tiga dimensi dengan kedalaman sesuai dengan banyaknya filter.

Gambar adalah contoh operasi konvolusi dengan input berukuran 5x5x1 dan filter berukuran 3x3x1.

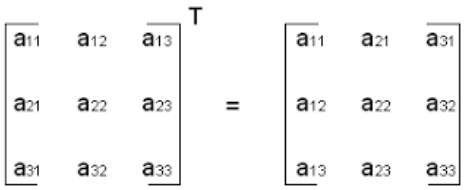


Gambar 2.6: Contoh Operasi Konvolusi. Image merupakan gambar masukan, convolved feature merupakan keluaran dari konvolusi.

Pada operasi konvolusi dengan input berukuran MxNxD dan filter berukuran M'xN'xD' dan banyak filter adalah K, maka kompleksitas operasi konvolusi adalah O(MM'NN'DK). Operasi ini memiliki kompleksitas yang paling tinggi diantara layer-layer lain.

2.3.3 Transpose Matriks

Transpose matriks dapat terjadi sebelum dilakukan perkalian matriks pada fully-connected layer. Operasi ini merupakan operasi membalik matriks terhadap diagonalnya. Pada matriks A dengan ukuran MxN, transpose dari A yaitu A' merupakan matriks yang tersusun dari elemen-elemen matriks A dimana A'ji = Aij untuk setiap bilangan bulat i dan j dimana 0 ¡= i ¡ M dan 0 ¡= j ¡ N. Gambar merupakan contoh operasi matriks transpose.

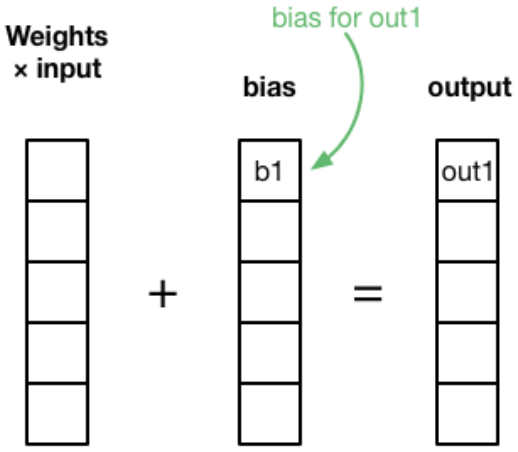


Gambar 2.7: Transpose matrix pada *Deep Learning inference* .

Matriks transpose memiliki kompleksitas O(MN).

2.3.4 Penjumlahan Matriks

Penjumlahan matriks dapat terjadi ketika melakukan penambahan bias terhadap hasil dari activation function suatu layer. Penjumlahan matriks dilakukan pada dua matriks berukuran sama. Pada dua matriks A dan B yang berukuran MxN, hasil penjumlahan matriks C merupakan matriks MxN dimana Cij = Aij + Bij untuk setiap bilangan bulat i dan j dimana 0 ¡= i ¡ M dan 0 ¡= j ¡ N. Gambar menunjukkan contoh operasi penjumlahan matriks.



Gambar 2.8: Penjumlahan matriks pada fully-connected layer saat menambahkan bias.

Pada kasus penambahan bias, lebar matriks biasanya adalah 1 atau bisa disebut sebagai vektor. Operasi ini memiliki kompleksitas O(MN), sama seperti matriks traspose.

2.4 Tensorflow

Tensorflow merupakan perangkat lunak *open-source* yang digunakan untuk komputasi numerik menggunakan representasi graf yang membentuk serangkaian operasi. Node pada graf merepresentasikan suatu operasi sedangkan edge mereperesentasikan tensor atau multidimensional array yang merupakan input atau output dari suatu operasi. Tensorflow dibuat oleh Google yang sejak awal digunakan untuk mendukung penelitian pada bidang *Machine Learning* dan Deep Learning. Saat ini Tensorflow sudah sangat dikenal sebagai salah satu *framework Machine Learning* dan *Deep Learning* yang memiliki banyak fitur dan mudah dioperasikan. Salah satu fitur dari Tensorflow yang saat ini cukup menarik banyak perhatian adalah adanya dukungan untuk perangkat *mobile* melalui *library* Tensorflow Mobile dan Tensorflow Lite yang digunakan pada penelitian ini.

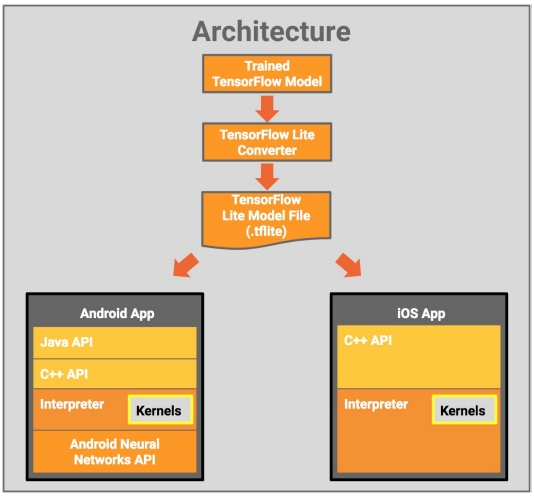
Tensorflow dan Tensorflow Lite diimplementasikan menggunakan bahasa C++ sehingga memungkinkan diterapkkannya OpenCL code. Kode sumber dari operasi-operasi *Deep Learning inference* pada Tensorflow Lite terpisah dari kode sumber operasi-operasi *Deep Learning inference* pada core Tensorflow. Modifikasi kode sumber Tensorflow Lite dengan menambahkan OpenCL code tidak akan mengubah behaviour dari program Tensorflow utama.

2.5 Tensorflow Lite

Tensorflow Lite merupakan salah satu library dari Tensorflow yang dapat digunakan untuk proses *Deep Learning inference* pada perangkat *mobile* . Tensorflow sebenarnya memiliki dua *library* yang dapat digunakan untuk melakukan Deep Learning *inference* pada perangkat mobile, yaitu Tensorflow Mobile dan Tensorflow Lite. Tensorflow Lite memungkinkan proses inference pada perangkat mobile hanya untuk dua jenis model Neural Network yaitu Convolutional Neural Network. Tensorflow Lite, yang dirilis pada November 14, 2017, merupakan versi perbaikan dari Tensorflow Mobile yang telah dirilis terlebih dahulu.

Dalam menjalankan Deep Learning inference, Tensorflow Lite menerima input sebuah file .tflite yang berisi model dari Neural Network yang telah dilatih. Model file ini sebenarnya merupakan graph operasi Tensorflow yang berisi serangkaian node dan edge dengan ekstensi .pb namun telah dikonversi ke .tflite menggunakan Tensorflow Lite Converter. Model file tersebut diload oleh C++ API untuk kemudian diteruskan ke interpreter. API ini tersedia untuk perangkat Android maupun iOS.

Interpreter mengeksekusi model menggunakan serangkaian kernel yang diload secara selektif, artinya hanya kernel yang berhubungan dengan operasi-operasi pada Model File saja yang diload. Ketika Tensorflow Lite dijalankan pada perangkat Android, ia akan menggunakan Android Neural Network API untuk meningkatkan performa inference. Namun tidak semua perangkat Android mendukung penggunaan Android Neural Network API. Gambar merupakan ilustrasi dari arsitektur Tensorfow Lite.



Gambar 2.9: Ilustrasi dari arsitektur Tensorflow Lite.

Dibandingkan dengan Tensorflow Mobile, Tensorflow Lite telah mendapatkan berbagai macam optimisasi di dalamnya, sehingga memiliki performa yang lebih baik dari Tensorflow Mobile. Beberapa bug yang ada pada Tensorflow Mobile juga telah diperbaiki pada Tensorlfow Lite. Selain itu, fitur-fitur yang ditawarkan Tensorflow Lite juga lebih banyak dibandingkan Tensorflow Mobile. Berikut adalah beberapa keunggulan Tensorflow Lite dibandingkan Tensorflow Mobile.

1. Memiliki fitur yang dapat memprune node dari operation graph yang tidak terpakai pada proses inference.
2. Memiliki kernel yang mendukung komputasi dalam 8-bit fixed point.
3. Optimasi performa untuk operasi floating point maupun fixed point.
4. Menggunakan pre-fused activation pada model Deep Learning.
5. Memuat kernel secara selektif. Hanya yang terkait operasi-operasi pada model saja yang dimuat.

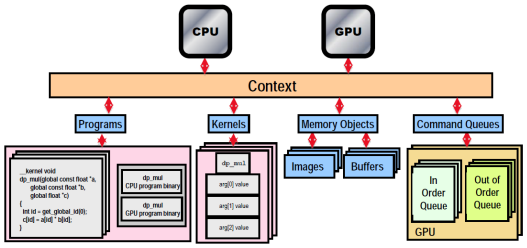
2.6 OpenCL

OpenCL merupakan antarmuka yang digunakan untuk pemrograman paralel pada prosesor yang berbeda-beda (CPU, GPU, dll) pada komputer, server, atau perangkat mobile. OpenCl dikenal dapat meningkatkan performa komputasi secara signifikan dengan menjalankan program secara paralel dan pada prosesor yang berbeda-beda. Pada OpenCL terdapat dua sisi program, yaitu host dan device. Program device merupakan program yang dijalankan pada processor device yang ditargetkan

(CPU, GPU, dll). Sementara itu program host berfungsi untuk mengatur jalannya program device, mulai dari menginisialisasi konteks, membuat program pada device, menyalin nilai variable dari host ke device memory, menjalankan kernel, dan mengambil nilai hasil komputasi dari device memory. Program host menggunakan bahasa C++ dengan beberapa ekstensi berupa fungsi dan tipe data khusus OpenCL.

Program yang dijalankan pada device terdiri dari satu atau lebih kernel. Kernel ditulis menggunakan bahasa C dengan sedikit perbedaan dari bahasa C biasa. Kernel pada OpenCl memiliki beberapa fungsi ekstensi dari bahasa C biasa, misalnya fungsi yang digunakan untuk mengambil worker ID dari suatu proses. Kernel ditulis sebagai string agar bisa digunakan oleh program host. Untuk dapat menjalankan program device, host harus terlebih dahulu membuat context. Context sendiri terbentuk dari satu atau lebih device. Contexts digunakan oleh OpenCL runtime untuk mengelola objek-objek seperti command-queues, memory, program, dan kernel. Context juga digunakan untuk mengeksekusi kernel-kernel pada satu atau lebih device yang telah terdefinisi pada context tersebut.

Setelah context terbentuk, host perlu membuat objek-objek lain seperti buffer dan command queue. Buffer merupakan bagian tertentu dari device memory yang digunakan untuk membaca atau menulis nilai yang diperlukan dalam proses kernel. Suatu buffer dapat bersifat read only, write only, atau read and write. Command queue merupakan antrian yang digunakan untuk menaruh perintah-perintah dari host kepada device. Contoh perintah adalah perintah untuk menjalankan kernel, perintah untuk menulis nilai ke buffer, dan perintah untuk membaca nilai dari buffer. Buffer dan command queue diinisialisasi pada bagian host. Gambar mengilustrasikan bagaimana OpenCL bekerja.



Gambar 2.10: Cara kerja OpenCL.

OpenCL dapat digunakan pada perangkat yang memiliki library OpenCL, biasanya bernama libOpenCl.so. Pada NVidia GPU, library dapat diperoleh dalam paket installasi CUDA. Sedangkan pada perangkat android, beberapa vendor GPU seperti Qualcomm menyertakan library OpenCL pada perangkat mereka. Library libOpenCl.so ini biasanya terletak pada direktori /system/vendor/lib/. Berikut adalah contoh program host dan device (kernel) pada OpenCL.

```
/* OpenCL Host Example */
void clHello() {
    cl_device_id device_id = NULL;
    cl_context context = NULL;
    cl_command_queue command_queue = NULL;
    cl_mem memobj = NULL;
    cl_program program = NULL;
    cl_kernel kernel = NULL;
    cl_platform_id platform_id = NULL;
    cl_uint ret_num_devices;
    cl_uint ret_num_platforms;
    cl_int ret;

    char string[MEM_SIZE];

    /* Get Platform and Device Info */
    ret = clGetPlatformIDs(1, &platform_id,
    &ret_num_platforms);
    ret = clGetDeviceIDs( platform_id,
    CL_DEVICE_TYPE_DEFAULT, 1, &device_id,
    &ret_num_devices);

    /* Create OpenCL context */
    context = clCreateContext( NULL, 1, &device_id,
    NULL, NULL, &ret);

    /* Create Command Queue */
    command_queue = clCreateCommandQueue(context,
    device_id, 0, &ret);

    /* Create Memory Buffer */
    memobj = clCreateBuffer(context,
    CL_MEM_READ_WRITE,MEM_SIZE * sizeof(char), NULL,
    &ret);

    /* Create Kernel Program from the source */
    program = clCreateProgramWithSource(context, 1,
```

```
    (const char **)&CLCL_HELLO,
    NULL, &ret);

    /* Build Kernel Program */
    ret = clBuildProgram(program, 1, &device_id, NULL,
    NULL, NULL);

    /* Create OpenCL Kernel */
    kernel = clCreateKernel(program, "hello", &ret);

    /* Set OpenCL Kernel Arguments */
    ret = clSetKernelArg(kernel, 0, sizeof(cl_mem),
    (void *)&memobj);

    /* Execute OpenCL Kernel */
    ret = clEnqueueTask(command_queue, kernel, 0,
    NULL,NULL);

    /* Copy results from the memory buffer */
    ret = clEnqueueReadBuffer(command_queue, memobj,
    CL_TRUE, 0,
    MEM_SIZE * sizeof(char),string, 0, NULL, NULL);

    /* Display Result */
    __android_log_print(ANDROID_LOG_DEBUG, "OpenCL",
    "Hello: %s", string);


    /* Finalization */
    ret = clFlush(command_queue);
    ret = clFinish(command_queue);
    ret = clReleaseKernel(kernel);
    ret = clReleaseProgram(program);
    ret = clReleaseMemObject(memobj);
    ret = clReleaseCommandQueue(command_queue);
    ret = clReleaseContext(context);
}
```

```
/* OpenCL kernel example */
const char *CLCL_HELLO =
    "__kernel void hello(__global char* string)\n"
    "{\n"
    "    string[0] = 'H';\n"
    "    string[1] = 'e';\n"
    "    string[2] = 'l';\n"
    "    string[3] = 'l';\n"
    "    string[4] = 'o';\n"
    "    string[5] = '\\0';\n"
    "}\n"
    "";
```

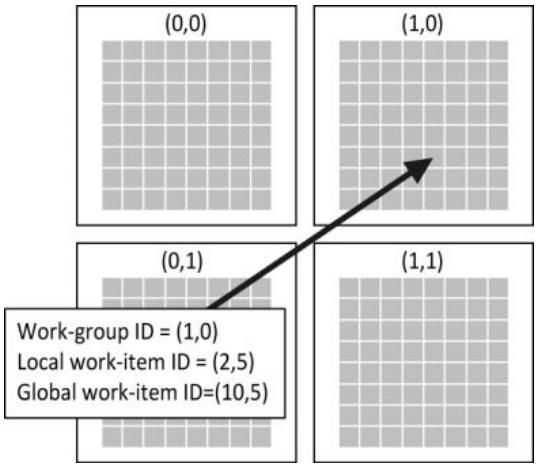
2.7 OpenCL Data Parallelism

OpenCL mendukung pemrograman paralel yang dapat dilakukan pada level data maupun level device. Ketika suatu program hanya dijalankan pada satu device saja, misalnya pada penelitian ini yang hanya menggunakan rGPU sebagai OpenCL device, paralelisasi tetap dapat dilakukan pada level data. Maksudnya, data yang diproses dipecah-pecah sehingga masing-masing unit pemrosesan hanya memproses sebagian dari data secara independen terhadap unit pemrosesan lainnya. Hal ini salah satunya dapat dilakukan pada operasi matriks, misalnya pada penjumlahan matriks, penambahan elemen-elemen matriks dapat dilakukan secara paralel dengan masing-masing unit pemrosesan hanya memproses elemen pada baris dan kolom yang sama saja.

Dalam OpenCL, unit pemrosesan ini disebut dengan work item. Setiap work item melakukan hal yang sama namun hanya datanya saja yang berbeda. Dengan kata lain, setiap work item menjalankan kernel yang sama. Masing-masing work item memiliki private memory. Setiap work item juga memiliki ID masing-masing yang dapat berupa ID secara global maupun ID work item itu dalam suatu grup tertentu. ID bilangan bulat positif yang dimulai dari 0.

Beberapa work item dapat membentuk work/local group. Proses yang terjadi pada suatu work group dapat melakukan sinkronisasi. Selain itu, work group memiliki memory tersendiri yang disebut local memory. Memory ini merupakan shared memory dimana setiap work item dalam grup dapat menggunakan local memory tersebut bersamaan. Work group juga memiliki ID seperti work item. Gambar

merupakan ilustrasi dari work item dan work group pada OpenCL.



Gambar 2.11: OpenCL work group dan work ite.

Saat suatu proses berjalan pada device, kita dapat mengambil ID dari work item atau work group yang menjalankan proses tersebut. Dengan demikian kita dapat mengatur suatu work item atau work group harus memproses bagian mana saja dari data. Misalnya dalam penjumlahan vektor, work item dengan ID i hanya memproses elemen ke i dari vektor-vektor yang dijumlahkan.

BAB 3
METODOLOGI

Pada bagian ini akan dijelaskan metodologi yang digunakan oleh Penulis pada saat melakukan penelitian. Metodologi ini merupakan tahap-tahap yang dilakukan dari awal hingga akhir penelitian.

3.1 Menentukan OpenCL dan Tensorflow Lite sebagai framework Penelitian

Pada penelitian ini Penulis memilih Tensorflow Lite sebagai *framework Deep Learning* pada perangkat mobile dengan beberapa alasan. Pertama, Tensorflow, termasuk Tensorflow Lite, merupakan perangkat lunak yang didukung oleh Google. Hingga saat ini Tensorflow masih sangat aktif dikembangkan, bisa dilihat dari frekuensi commit pada repository GitHub Tensorflow. Kedua, antarmuka Tensorflow Lite mudah dipahami dan digunakan. Ketiga, Tensorflow Lite memiliki lebih banyak fitur dibandingkan library lain seperti dukungan untuk operasi 8-bit, graph pruning, dan lain-lain. Keempat, jika dibandingkan dengan framework lain seperti Torch atau Caffe, Tensorflow Lite memiliki popularitas yang lebih tinggi dilihat dari statistik fork, star, dan watch pada GitHub.

Kemudian, penulis memilih OpenCL sebagai antarmuka pemrograman GPU adalah karena OpenCL sebenarnya tidak hanya dapat digunakan untuk GPU namun juga perangkat-perangkat lain seperti FPGA atau DSP. Dengan demikian program OpenCL yang dibuat dapat dimodifikasi dengan mudah agar dapat digunakan ulang untuk keperluan eksperimen pada perangkat lain. Selain itu, OpenCL menggunakan bahasa C/C++ dengan beberapa ekstensi yang mudah dipahami. Penggunaan C/C++ mendukung implementasi OpenCL pada Tensorflow.

3.2 Mempelajari kode sumber Tensorflow Lite

Penulis mempelajari kode sumber Tensorflow Lite tertuama pada gabian implementasi kernel yang berisi operasi-operasi *Deep Learning inference* . Kode sumber Tensorflow Lite ditulis dalam bahasa C dan C++. Berbeda dengan Tenosrflow Mobile dimana kode untuk operasi-operasi tersebut menyatu dengan Tensorflow core, pada Tensorflow Lite terdapat abstraksi tersendiri untuk opperasi-operasi *Deep Learning*

inference . Vendor dari perangkat mobile dapat memberikan implementasi kernel *Deep Learning inference* yang sudah dioptimisasi. Jika tidak diberikan, Tensorflow Lite akan menggunakan implementasi default untuk CPU.

3.3 Mempelajari OpenCL

Penulis mempelajari mengenai konsep dari OpenCL, yaitu bagaimana OpenCL bekerja. Kemudian, penulis mempelajari cara menggunakan OpenCL dimulai dari mempelajari sintaks-sintaksnya. Terdapat dua bagian yang perlu dipelajari yaitu bagian host dan device. Kedua bagian tersebut menggunakan bahasa C/C++.

Pada bagian host penulis mempelajari bagaimana cara menginisialisasi program OpenCL termasuk cara membuat kernel, membuat program, membuat buffer, membuat queue, dan menjalankan kernel melalui queue. Penulis juga mempelajari tipe-tipe data yang dapat digunakan pada OpenCL. Sedangkan pada bagian device atau kernel penulis mempelajari bagaimana cara membuat fungsi pada kernel, mengambil ID dari work item atau work group, serta cara melakukan sinkronisasi. Dalam mempelajari sintaks host maupun device penulis sudah terlebih dahulu memahami bahasa C/C++ sehingga tinggal mempelajari ekstensi-ekstensi untuk OpenCL saja.

Setelah mempelajari sintaks, penulis mempelajari metode pemrograman paralel pada OpenCL, misalnya teknik-teknik apa saja yang dapat diterapkan agar program , terutama dalam program operasi-operasi matriks. Penulis juga mempelajari bagaimana cara meng-compile, menjalankan program OpenCL, serta cara menggunakannya pada perangkat mobile.

3.4 Mengimplementasikan OpenCL accelerator untuk Tensorflow Lite

Implementasi accelerator dimulai dengan mengintegrasikan OpenCL dengan Tensorflow Lite. Ini dapat dilakukan dengan mengambil terlebih dahulu library OpenCL dari perangkat mobile yang akan digunakan untuk eksperimen. Library ini dapat diambil dari direktori /system/vendor/lib dengan mengguanakan adb pull. Setelah diperoleh, libaray ini ditempatkan pada direktori library dari Android NDK yang digunakan sehingga dapat terdeteksi oleh compiler ketika mem-build Tensorflow Lite dengan mengikutsertakan nama library pada flag.

Kemudian, diperlukan pula C++ headers untuk dapat memanggil fungsi-fungsi pada OpenCL. Untuk perangkat mobile dapat menggunakan header dari ARM. Header ini diletakkan pada direktori kernel Tensorflow Lite dan edit build file

agar Tensorflow Lite mengikutsertakan direktori header tersebut ketika mengcompile. Setelah OpenCL terintegrasi dengan Tensorflow Lite dan dapat berjalan pada perangkat android, penulis memulai implementasi OpenCL code untuk operasi-operasi *Deep Learning inference* yang ada pada direktori kernel Tensorflow Lite. Operasi-operasi yang diimplementasikan menggunakan OpenCL antara lain adalah perkalian matriks, konvolusi matriks, transpose matriks, dan penjumlahan matriks.

3.5 Melakukan eksperimen terhadap hasil implementasi

Setelah implementasi perkalian matriks, konvolusi matriks, transpose matriks, dan penjumlahan matriks menggunakan OpenCL pada Tensorflow Lite selesai dilakukan, penulis mencoba melakukan eksperimen untuk membandingkan kecepatan masing-masing operasi pada proses inference.

Eksperimen dilakukan pada perangkat android pribadi dengan menggunakan aplikasi Tensorflow Lite Demo app yang merupakan sebuah aplikasi image recognition. Performa dari operasi-operasi matriks saat inference diukur menggunakan wall time yang diimplementasikan sendiri oleh penulis menggunakan C++. Penulis menggunakan beberapa model Convolutional Neural Netowork untuk aplikasi ini yaitu Inception, LeNet, dan MobileNet. Saat inference terjadi, model-model tersebut melibatkan semua operasi matriks yang telah diimplementasikan menggunakan OpenCL.

3.6 Mengoptimalkan implementasi sebelumnya

Setelah mengevaluasi hasil eksperimen terhadap operasi-operasi yang diimplementasikan menggunakan OpenCL dan membandingkannya dengan implementasi asli dari Tensorflow yang hanya menggunakan CPU, penulis menemukan bahwa kecepatan operasi-operasi tersebut sudah sangat baik untuk matriks-matriks yang cukup besar. Namun, penulis melihat bahwa code OpenCL dari operasi-operasi tersebut terutama pada bagian kernel (device code) masih dapat dioptimalkan, misalnya dengan mengubah skema paralelisasi pemrosesan matriks. Sehingga pada tahap ini penulis pun mencoba melakukan optimisasi dari implementasi OpenCL yang telah dibuat dengan harapan adanya peningkatan performa.

3.7 Melakukan eksperimen terhadap hasil implementasi yang sudah dioptimalkan

Eksperimen kembali dilakukan pada perangkat yang sama dan skenario yang sama untuk melihat perbedaan performa dari implementasi sebelumnya. Performa yang diperoleh dari hasil eksperimen ini kemudian juga dibandingkan dengan performa implementasi asli operasi-operasi *Deep Learning inference* dari Tensorflow Lite.

3.8 Mengevaluasi dan Membandingkan Performa Proses *Inference*

Evaluasi dari hasil optimisasi dilakukan menggunakan Tensorflow GPU dan Tensorflow CPU yang di-*compile* melalui kode sumber. Model CNN yang digunakan untuk melakukan evaluasi dalah Inception V1. Penulis mengamati performa, efisiensi energi, serta penggunaan memori pada setiap proses *inference* pada skenario yang berbeda-beda. Selain melakukan evaluasi pada keseluruhan proses *inference* , evaluasi juga dilakukan pada setiap operasi pada *operation graph* . Setelah performa masing-masing proses *inference* dievaluasi, penulis membandingkan hasilnya dan melakukan analisis terhadap hasil-hasil tersebut.

3.9 Melaporkan hasil penelitian

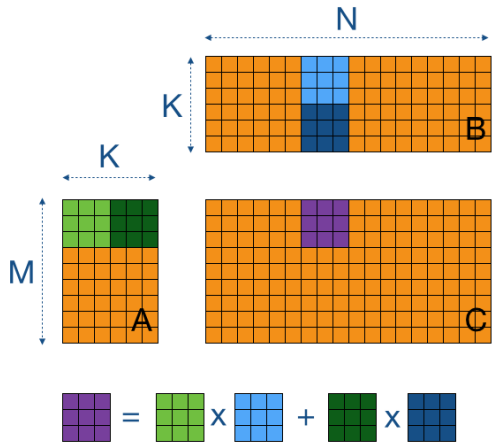
Hasil dari penelitian ini dilaporkan dalam bentuk Tugas Akhir dan dipresentasikan saat Sidang Tugas Akhir.

BAB 4
IMPLEMENTASI

Bagian ini berisi penjelasan mengenai implementasi program-program yang terkait dengan penelitian. Program tersebut meliputi perkalian matriks, kovolusi matriks, transpose matriks, serta penjumlahan matriks yang diimplementasikan menggunakan OpenCL dan diintegrasikan ke Tensorflow Lite.

4.1 OpenCL Matriks Multiplication

Implementasi perkalian matriks menggunakan blocking matrix multiplication. Kedua matriks yang dikalikan dibagi ke dalam beberapa block yang sama dimana satu block memiliki ukuran 32x32. Perhatikan pada gambar , terdapat dua matriks A dan B yang masing-masing berukuran MxK dan KxN. Dengan demikian perkalian keduanya menghasilkan matriks C yang berukuran MxN. Blok pada matriks C yang berwarna ungu diperoleh dari dua tahap perkalian matriks, yaitu perkalian antara blok hijau muda dan blok biru muda pada matriks A dan B kemudian ditambahkan dengan perkalian antara blok hijau tua dan blok biru tua pada matriks A dan B.



Gambar 4.1: Perkalian matriks per blok.

Motivasi dari penggunaan blok pada perkalian matriks adalah untuk meningkatkan efisiensi penggunaan cache pada GPU. Perhatikan bahwa untuk mendapatkan suatu elemen baris ke i dan kolom ke j dari matriks, baris ke i dari matriks A dan kolom ke j dari akan diload ke memory dan cache. Kemudian untuk mendapatkan suatu elemen baris ke i+1 dan kolom ke j dari matriks, kolom ke j dari matriks

B kembali diload ke memory. Kolom ke j dari matriks B ini selalu digunakan ulang dalam menghitung elemen kolom ke j pada matriks C.

Untuk matriks berukuran besar, besar kemungkinan bahwa kolom tersebut sudah terhapus dari cache karena sebelum memproses kolom itu lagi, cache terlebih dahulu digunakan untuk menyimpan kolom-kolom lain karena sudah banyak perkalian yang dilakukan. Sedangkan ketika matriks berukuran kecil, peluang kolom tersebut masih berada di cache lebih tinggi, sehingga komputasi menggunakan kolom tersebut menjadi lebih cepat. Inilah mengapa matriks dibagi ke dalam beberapa bagian yang lebih kecil untuk dikalikan secara bertahap agar frekuensi penggunaan cache lebih tinggi sehingga komputasi lebih cepat.

Perkalian per blok ini dapat diimplementasikan dalam OpenCL kernel seperti berikut. Iterasi dilakukan sebanyak K/blocksize kali. Pada setiap iterasi, perkalian per blok dilakukan untuk 32 kolom dari matriks A (semua baris) dan 32 baris dari matriks B (semua kolom). Ingat bahwa ukuran blok adalah 32x32.

```
__kernel void matrixVectorMul(__global float* C,
    const __global float* A,
    const __global float* B,
    int K, int M, int N) {

        // Local row ID (max: 32)
    const int row = get_local_id(0);
    // Local col ID (max: 32)
    const int col = get_local_id(1);
    // Row ID of C (0..M)
    const int globalRow = 32*get_group_id(0) + row;
    // Col ID of C (0..N)
    const int globalCol = 32*get_group_id(1) + col;

    __local float Asub[32][32];
    __local float Bsub[32][32];

    float acc = 0.0;

    const int numTiles = ((K-1)/32)+1;
    for (int t=0; t<numTiles; t++) {

        const int tiledRow = 32*t + row;
```

```
        const int tiledCol = 32*t + col;
        if((tiledCol < K) && (globalRow < M)) {
            Asub[col][row] = A[globalRow*K + tiledCol];
        }
        else {
            Asub[col][row] = 0.0;
        }
        if((tiledRow < K) && (globalCol < N)) {
            Bsub[col][row] = B[globalCol*K + tiledRow];
        }
        else {
            Bsub[col][row] = 0.0;
        }

        barrier(CLK_LOCAL_MEM_FENCE);

        for (int k=0; k<32; k++) {
            acc += Asub[k][row] * Bsub[col][k];
        }

        barrier(CLK_LOCAL_MEM_FENCE);
    }

    if((globalRow < M) && (globalCol < N)) {
        C[globalCol*M + globalRow] = acc;
    }
}
```

4.2 OpenCL Matriks Convolution

4.3 OpenCL Matriks Transpose

Skema paralelisasi yang digunakan pada operasi transpose ini adalah masing-masing work item memproses hanya satu elemen dari matriks, sehingga diperlukan sebanyak MxN work-item untuk matriks berukuran MxN. Jika matriks yang di-traspose adalah matriks A dan hasil dari transpose adalah matriks At, maka proses yang dilakukan setiap work-item dengan index (i,j) adalah mengassign nilai matriks

At pada index (j,i) dengan nilai matriks A pada index (i,j). Gambar merupapakan ilustrasi dari skema paralelisasi yang digunakan.

Berikut adalah OpenCL code yang digunakan.

```
__kernel void transpose(__global float* input,
    __global float* output, int rows, int cols) {
    int gid = get_global_id(0);

    if(gid < rows*cols) {
        int i = gid/cols;
        int j = gid%cols;
        const float in_value = input[gid];
        output[j*rows + i] = in_value;
    }
}
```

4.4 OpenCL Matriks Addition

Skema paralelisasi yang digunakan pada operasi transpose ini adalah masing-masing work item memproses hanya satu elemen dari matriks, sehingga diperlukan sebanyak MxN work-item untuk matriks berukuran MxN. Jika matriks yang di-traspose adalah matriks A dan hasil dari transpose adalah matriks At, maka proses yang dilakukan setiap work-item dengan index (i,j) adalah mengassign nilai matriks At pada index (j,i) dengan nilai matriks A pada index (i,j). Gambar merupapakan ilustrasi dari skema paralelisasi yang digunakan.

Berikut adalah OpenCL code yang digunakan.

```
__kernel void transpose(__global float* inputA,
    __global float* inputB, __global float* output,
    int rows, int cols) {
    int gid = get_global_id(0);

    if(gid < rows*cols) {
        const float in_valueA = inputA[gid];
        const float in_valueB = inputB[gid];
        output[gid] = in_valueA + in_valueB;
    }
}
```

BAB 5
EKSPERIMEN DAN ANALISIS

Bagian ini bersisi pelaksanaan dan hasil eksperimen serta analisis dari hasil tersebut.

5.1 Mengubah Tampilan Teks

5.2 Memberikan Catatan

5.3 Menambah Isi Daftar Isi

5.4 Memasukan PDF

5.5 Membuat Perintah Baru

BAB 6
KESIMPULAN DAN SARAN

@todo
tambahkan kata-kata pengantar bab 6 disini

6.1 Kesimpulan

6.2 Saran

DAFTAR REFERENSI

[1] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436444. https://doi.org/10.1038/nature14539

[2] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Rabinovich, A. (2015). Going deeper with convolutions. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. https://doi.org/10.1109/cvpr.2015.7298594

[3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 8490. https://doi.org/10.1145/3065386

[4] Liu, S., & Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. In 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR). IEEE. https://doi.org/10.1109/acpr.2015.7486599

[5] Russakovsky, O., Deng, J., Su, H. et al. Int J Comput Vis (2015) 115: 211. https://doi.org/10.1007/s11263-015-0816-y

[6] O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints.

[7] Martn Abadi, dkk.. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] Saving and Restoring — TensorFlow. (n.d.). Retrieved December 19, 2017, from https://www.tensorflow.org/programmers_guide/saved_model

[9] A Tool Developer's Guide to TensorFlow Model Files — TensorFlow. (n.d.). Retrieved December 19, 2017, from https://www.tensorflow.org/extend/tool_developers/#freezing

[10] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16), Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. JMLR.org 2849-2858.

[11] Chandra, V., Lai, L., & Suda, N. (2017). Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations. CoRR, abs/1703.03073.

[12] Latifi Oskouei, S. S., Golestani, H., Hashemi, M., & Ghiasi, S. (2016). CNNdroid. In Proceedings of the 2016 ACM on Multimedia Conference - MM 16. ACM Press. https://doi.org/10.1145/2964284.2973801

[13] How to Quantize Neural Networks with TensorFlow — TensorFlow. (n.d.). Retrieved December 20, 2017, from https://www.tensorflow.org/performance/quantization

[14] B. (2017, November 29). BVLC/caffe. Retrieved December 20, 2017, from https://github.com/BVLC/caffe

[15] Jaderberg, M., Vedaldi, A., & Zisserman, A. (2014). Speeding up Convolutional Neural Networks with Low Rank Expansions. In Proceedings of the British Machine Vision Conference 2014. British Machine Vision Association. https://doi.org/10.5244/c.28.88

[16] Huang, S., Xiao, S., & Feng, W. (2009). On the energy efficiency of graphics processing units for scientific computing. In 2009 IEEE International Symposium on Parallel & Distributed Processing. IEEE. https://doi.org/10.1109/ipdps.2009.5160980

LAMPIRAN

LAMPIRAN 1