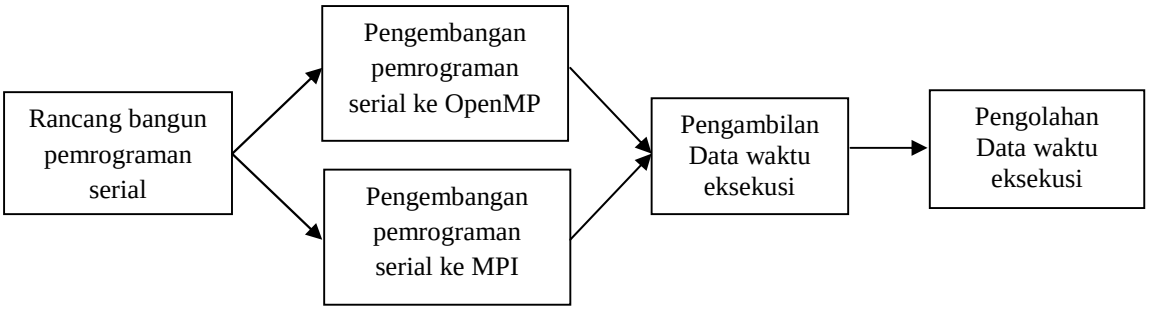
BAB III
METODE PENELITIAN

Penelitian implementasi pemrograman paralel dalam deteksi tepi menggunakan metode operator Sobel dibuat dengan menggunakan bahasa pemrograman C++. Metode penelitian yang digunakan adalah metode eksperimen, dimana kode-kode program yang telah dibuat dijalankan untuk diproses oleh prosesor *multicore.*

Untuk mencapai tujuan penelitian, diperlukan beberapa tahapan penelitian. Tahapan-tahapan tersebut dapat disimpulkan dalam diagram blok seperti terlihat pada Gambar 3.1.



Gambar 3.1 Diagram blok metode penelitian

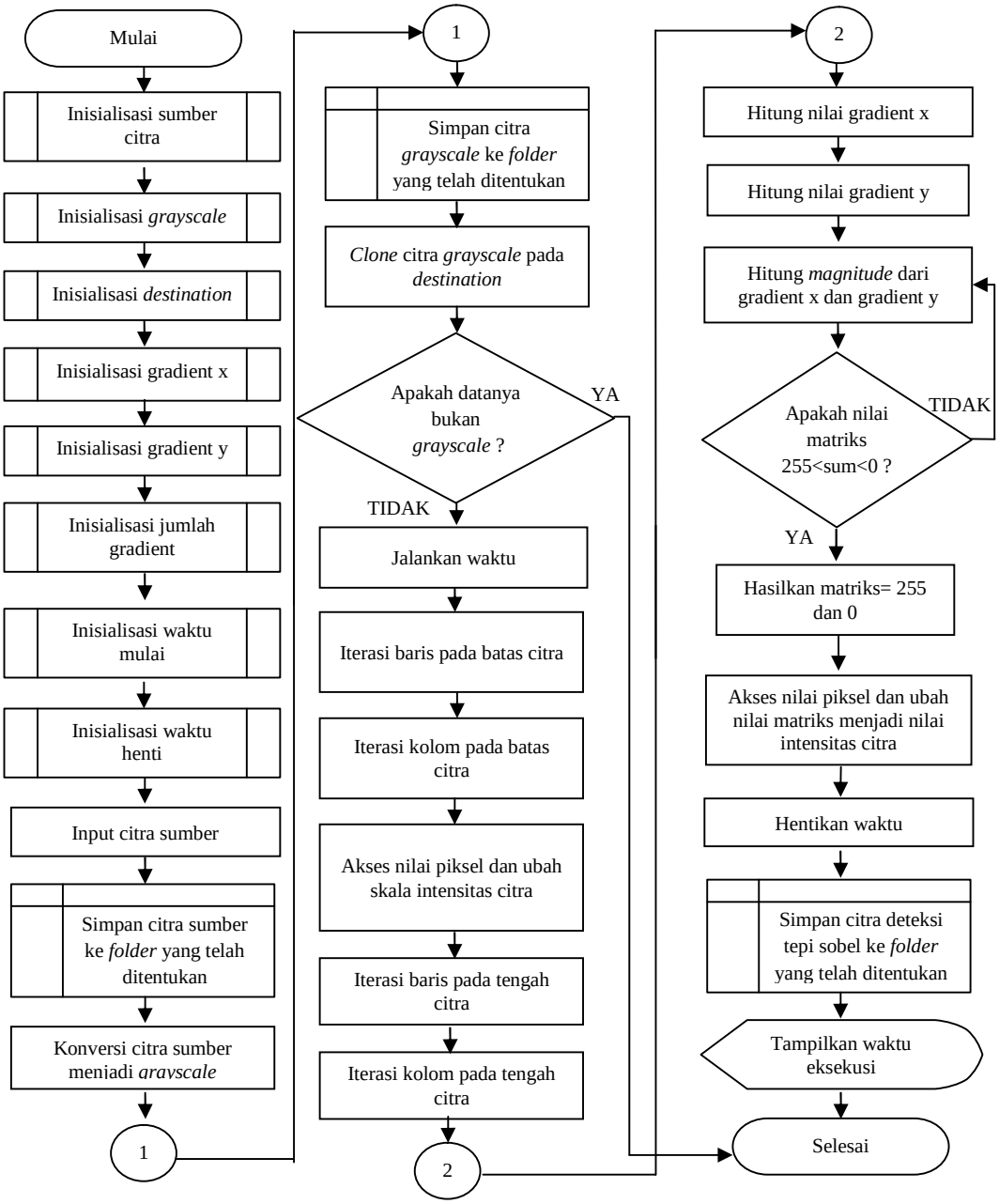
Diagram blok dibuat untuk menampilkan langkah-langkah yang dilakukan dalam melakukan penelitian ini. Tahapan pertama adalah merancang dan membangun pemrograman serial deteksi tepi menggunakan metode operator Sobel, yaitu dengan cara membuat *flowchart* pemrograman serial, kemudian *flowchart* tersebut diterjemahkan ke dalam kode-kode program bahasa C++.

Tahapan kedua adalah pengembangan pemrograman serial ke pemrograman paralel. Ada dua pustaka pemrograman paralel yang digunakan untuk pengembangan pemrograman serial yaitu OpenMP dan MPI. Tahapan ketiga adalah pengambilan data waktu eksekusi dengan cara mengeksekusi kode program yang telah dibuat secara serial dan paralel, kemudian dicatat waktu eksekusinya.

Tahapan keempat adalah pengolahan data waktu eksekusi dengan cara menghitung nilai *speedup.* Untuk mendapatkan nilai *speedup* dibutuhkan dua nilai yaitu rata-rata waktu eksekusi pemrograman serial dan rata-rata waktu eksekusi pemrograman paralel menggunakan pustaka OpenMP dan MPI pada bahasa pemrograman C++.

3.1 Rancang bangun pemrograman serial

Program serial deteksi tepi operator Sobel dirancang menggunakan *flowchart* (diagram alir). *Flowchart* adalah penggambaran secara grafis dari langkah-langkah dan urutan prosedur dari suatu program. *Flowchart* memberikan kemudahan *programmer* untuk menulis kode program. *Flowchart* sangatlah penting sebagai tahap awal dari perancangan sebuah algoritma, karena *flowchart* menggambarkan langkah-langkah program yang akan dibuat. Setelah perancangan *flowchart* selanjutnya adalah penulisan kode program untuk menjelaskan *flowchart*. Adapun *flowchart* pemrogaman serial deteksi tepi menggunakan operator Sobel seperti terlihat pada Gambar 3.2.



Gambar 3.2 *Flowchart* pemrogaman serial deteksi tepi menggunakan operator Sobel

Pada pemrograman serial deteksi tepi menggunakan operator Sobel, program dimulai dengan inisialisasi data-data yang diperlukan dalam deteksi tepi mulai dari inisialisasi sumber citra yang digunakan hingga inisialisasi waktu ekekusi. Kode program inisialisasi secara berturut-turut yaitu:

```
.....
int main ()
{
   Mat src, grey, dst;
   int gx, gy, sum;
   clock_t start, finish;
.....
```

Langkah selanjutnya adalah peng-*input*-an citra digital dari sebuah *file* yang dideteksi. *Input* dari program yang dibuat adalah *file* citra berwarna (RGB) dengan jenis *file* citra *windows bitmap* (BMP). Fungsi *imread* memuat citra dari *folder* yang ditentukan. Kode program yang digunakan yaitu:

```
.....
src= imread("E:/tigaout/Debug/view_sea.bmp");
.....
```

Citra digital tersebut disimpan sebagai *output* citra asli. Fungsi *imwrite* digunakan untuk menyimpan citra asli ke *folder* yang ditentukan yaitu *folder* Serial. Kode program yang digunakan yaitu:

```
.....
imwrite("E:/tigaout/Debug/Serial/Citra Asli.bmp", src);
.....
```

Citra yang dibaca adalah citra warna (RGB) sehingga citra perlu diubah ke dalam citra *grayscale*. Pengubahan citra asli (citra warna) menjadi citra *grayscale* menggunakan perintah:

```
.....
cvtColor(src, grey, CV_BGR2GRAY);
.....
```

Citra *grayscale* tersebut disimpan sebagai *output*. Fungsi *imwrite* digunakan untuk menyimpan citra *grayscale* ke *folder* yang ditentukan yaitu *folder* Serial. Kode program yang digunakan yaitu:

```
.....
imwrite("E:/tigaout/Debug/Serial/Grayscale.bmp", grey );
.....
```

Langkah selanjutnya adalah membuat salinan (*clone*) citra *grayscale* untuk objek baru yaitu *destination*. Jika data yang di-*clone* bukan merupakan data *grayscale* maka program akan selesai. Kode program yang digunakan yaitu:

```
.........
dst = grey.clone ();
if ( !grey.data )
{
    return -1;
}
.........
```

Setelah proses *clone* citra *grayscale* selesai, maka langkah selanjutnya adalah menjalankan waktu eksekusi dan melakukan iterasi pada semua piksel yaitu iterasi setiap baris serta iterasi setiap kolom dari keseluruhan piksel citra. Kode program yang digunakan untuk langkah ini yaitu:

```
.........
start = clock ();
for (int y = 0; y < grey.rows; y++)
    for (int x = 0; x < grey.cols; x++)
.........
```

Kode program di atas hanya melakukan iterasi dari indeks 0 sampai indeks 1. Hal ini dilakukan untuk iterasi pada perbatasan (*border*) citra yaitu hanya pada elemen pertama dan terakhir dari semua piksel.

Langkah selanjutnya adalah kebalikan dari iterasi yang dilakukan pada perbatasan citra. Ini adalah teknik sederhana dimana mengabaikan piksel di perbatasan citra untuk melengkapi iterasi sebelumnya agar tercipta iterasi secara keseluruhan pada semua piksel sebuah citra. Hal ini dapat dilakukan dengan menggunakan algoritma berikut :

```
.........
for (int y = 1; y < grey.rows - 1; y++){
    for (int x = 1; x < grey.cols - 1; x++){
.........
```

Karena iterasi dari indeks 1 sampai dengan indeks terakhir - 1, maka iterasi tidak dapat dievaluasi di perbatasan citra. Hal itu akan menghindari elemen pertama dan terakhir pada citra. Jadi pada *loop* ini, semua yang akan diiterasi adalah piksel-piksel selain dari piksel yang ada pada perbatasan citra.

Setelah melakukan iterasi pada semua piksel, langkah selanjutnya adalah menghitung nilai gradien dalam arah x dan menghitung nilai gradien dalam arah y. Kode program yang digunakan yaitu:

```
.....
gx = xGradient(grey, x, y);
gy = yGradient(grey, x, y);
.....
```

Kode program yang digunakan untuk menghasilkan nilai gradien dengan arah x dan y menggunakan Persamaan 2.1 yang ditulis ke dalam bahasa pemrograman C++. Dimana gx dan gy memanggil fungsi xGradient dan yGradient untuk menghasilkan gradien pada arah horizontal dan vertikal. Kode program fungsi xGradient dan yGradient yaitu:

```
int xGradient(Mat image, int x, int y)
{
   return image.at<uchar>(y-1, x-1) +
               2*image.at<uchar>(y, x-1) +
                image.at<uchar>(y+1, x-1) -
                 image.at<uchar>(y-1, x+1) -
                  2*image.at<uchar>(y, x+1) -
                   image.at<uchar>(y+1, x+1);
}

int yGradient(Mat image, int x, int y)
{
   return image.at<uchar>(y-1, x-1) +
               2*image.at<uchar>(y-1, x) +
                image.at<uchar>(y-1, x+1) -
                 image.at<uchar>(y+1, x-1) -
                  2*image.at<uchar>(y+1, x) -
                   image.at<uchar>(y+1, x+1);
}
```

Setelah nilai gradient x dan gradient y dihitung, langkah selanjutnya adalah menghitung *magnitude* dari gradient x dan gradient y. Kode program yang digunakan yaitu:

```
.....
sum = abs(gx) + abs(gy);
.....
```

Kode program yang digunakan untuk menghasilkan *magnitude* dari gradient dihitung menggunakan Persamaan 2.3 yang ditulis ke dalam bahasa pemrograman C++.

Langkah selanjutnya adalah mengubah nilai matriks yang didapat dari hasil *magnitude* gradient menjadi nilai intensitas citra dan dihasilkan garis tepi suatu citra digital. Kode program yang digunakan yaitu:

```
.....
sum = sum > 255 ? 255:sum;
sum = sum < 0 ? 0 : sum;
dst.at<uchar>(y,x) = sum;
.....
```

Setelah dihasilkan garis tepi, langkah selanjutnya adalah menghentikan waktu eksekusi. Kode program yang digunakan yaitu:

```
.........
finish = clock();
.........
```

Setelah waktu eksekusi dihentikan, langkah selanjutnya adalah menyimpan citra hasil deteksi tepi Sobel sebagai *output* dari program yang telah dieksekusi dalam bentuk *file* berekstensi bmp ke *folder* yang telah ditentukan yaitu *folder* Serial. Kode program yang digunakan untuk langkah ini yaitu:

```
.........
imwrite( "E:/tigaout/Debug/Serial/output deteksi tepi sobel.bmp", dst);
.........
```

Setelah hasil deteksi tepi Sobel tersimpan, langkah selanjutnya adalah menampilkan waktu eksekusi dari pemrosesan serial deteksi Sobel dan penghentian program. Kode program yang digunakan untuk langkah ini yaitu:

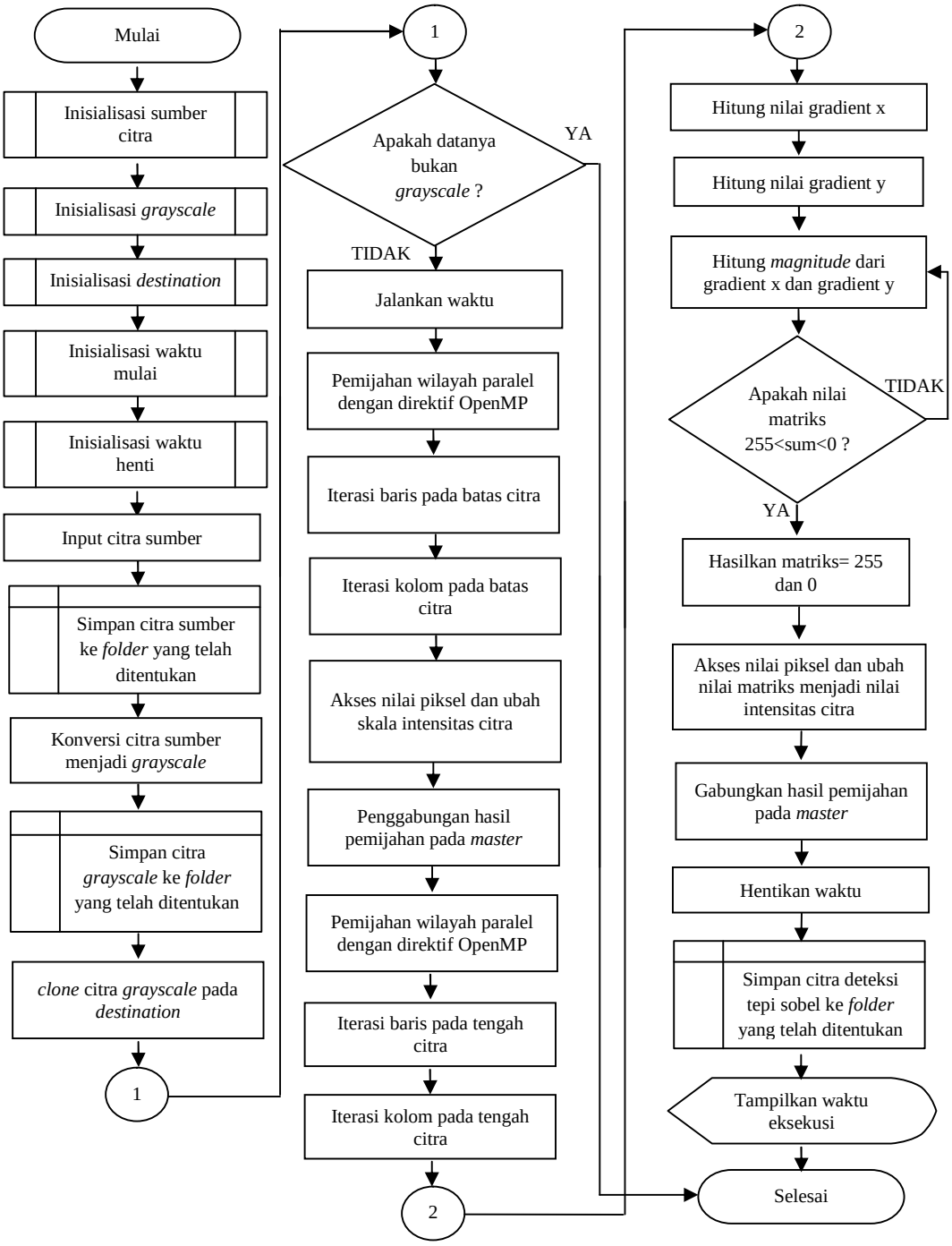
```
    .........
    cout << "waktu eksekusi serial adalah : " << float(finish-start)/CLOCKS_PER_SEC << " detik" << endl;
    return 0;
}
```

3.2 Pengembangan Pemrograman Serial ke Pemrograman Paralel

Pada subbab ini akan dibahas tentang pengembangan pemrograman serial ke OpenMP dan MPI.

3.2.1 Pengembangan Pemrograman Serial Ke OpenMP

OpenMP merupakan antarmuka pemrograman aplikasi yang mendukung *shared memory*. OpenMP mengimplementasi *multithreading*. Bagian kode yang dijalankan secara paralel ditandai sesuai dengan direktif *compiler*. *Flowchart* pemrograman paralel dengan OpenMP dirancang untuk memudahkan *programmer* memparalelkan program serial yang telah dibuat. Adapun *flowchart* pemrogaman paralel dengan OpenMP pada deteksi tepi menggunakan operator Sobel seperti terlihat pada Gambar 3.3.



Gambar 3.3 *Flowchart* pemrogaman paralel OpenMP deteksi tepi menggunakan operator Sobel

Model dari pemrograman OpenMP untuk pemijahan wilayah agar masing-masing *thread* melakukan iterasi baris dan kolom pada batas citra, dapat dilihat pada Gambar 3.4.

Gambar 3.4 Model pemrogaman OpenMP untuk iterasi baris dan kolom pada batas citra

Paralelisasi data dalam *loop* adalah bentuk paling umum dari paralelisasi yang digunakan dalam OpenMP dengan cara menggunakan direktif *compiler*. Kode program yang digunakan untuk langkah ini yaitu:

```
.....
#pragma omp parallel for
        for(int y = 0; y < grey.rows; y++)
                for(int x = 0; x < grey.cols; x++)
.....
```

Looping sering dapat diparalelkan untuk meningkatkan kinerja. Untuk membuat proses iterasi baris dan kolom pada batas citra agar bekerja secara paralel maka pada bagian iterasi ditambahkan *pragma omp paralel for* untuk membangkitkan beberapa *thread* agar iterasi pada batas citra dapat dikerjakan secara bersama-sama. Ketika semua *thread* telah melakukan *task*-nya sampai selesai, tahap selanjutnya adalah menggabungkan kembali hasil dari eksekusi masing-masing *thread* pada *master thread*.

Langkah selanjutnya adalah membuat proses iterasi pada tengah citra dan proses fungsi Sobel agar bekerja secara paralel. Adapun model pemrogaman OpenMP untuk iterasi baris dan kolom pada tengah citra dan proses fungsi Sobel seperti terlihat pada Gambar 3.5.

Gambar 3.5 Model pemrogaman paralel OpenMP untuk iterasi baris dan kolom pada tengah citra dan proses fungsi sobel

Dalam tahap ini ada dua *task* yang dikerjakan secara paralel yaitu iterasi baris dan kolom pada tengah citra dan proses fungsi Sobel. Setiap *task* berjalan di masing-masing *thread* dengan menggunakan direktif *compiler* OpenMP. Kode program yang digunakan untuk langkah ini yaitu:

```
.............
   #pragma omp parallel for
   for(int y = 1; y < grey.rows - 1; y++)
   {
      for(int x = 1; x < grey.cols - 1; x++)
     {
           int gx = xGradient(grey, x, y);
           int gy = yGradient(grey, x, y);
           int sum = abs(gx) + abs(gy);
           sum = sum > 255 ? 255:sum;
           sum = sum < 0 ? 0 : sum;
           dst.at<uchar>(y,x) = sum;
      }
   }
```

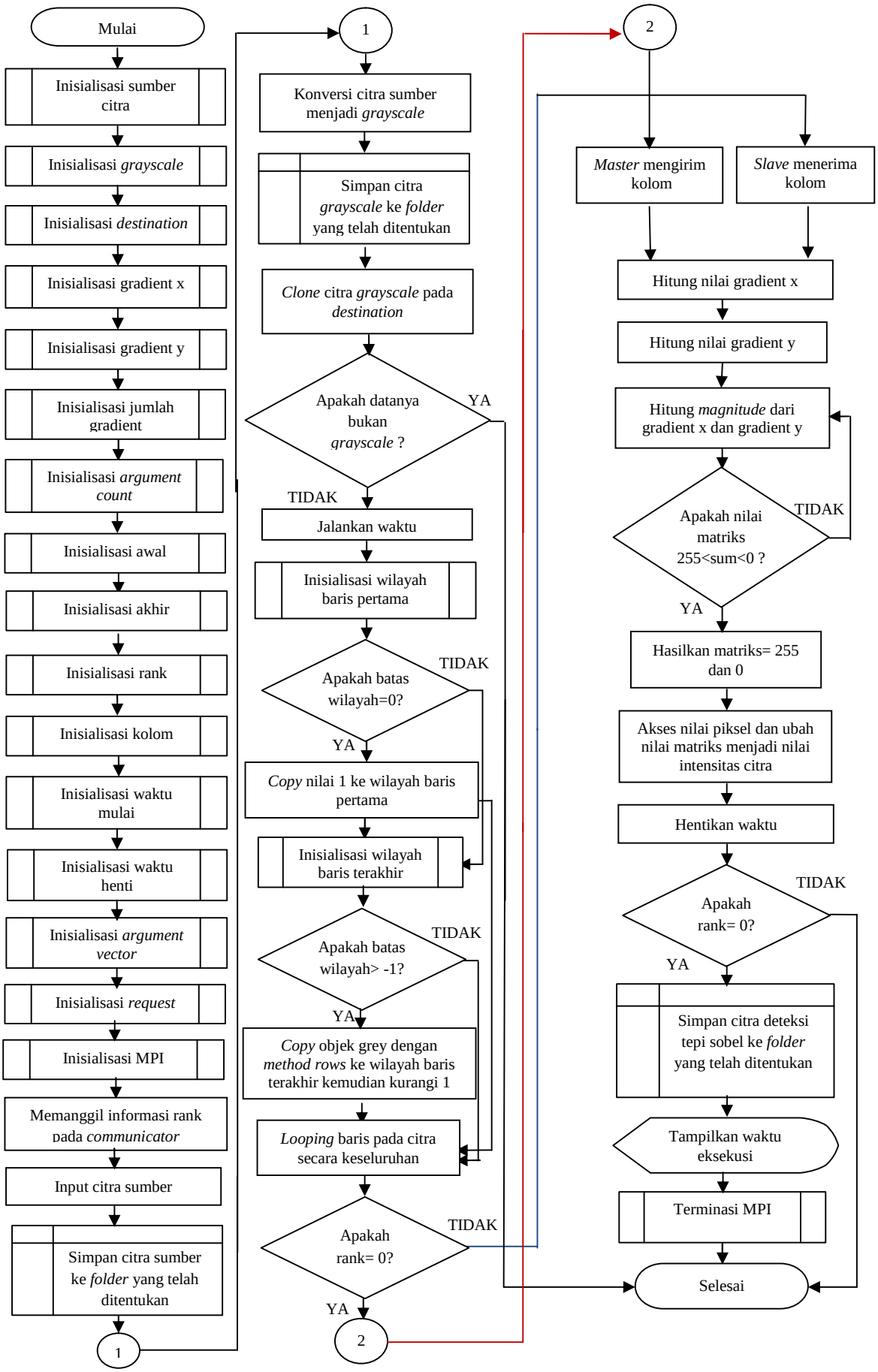
Untuk membuat proses iterasi baris dan kolom pada tengah citra dan fungsi Sobel agar bekerja secara paralel maka pada kode *looping* dan fungsi Sobel ditambahkan *pragma omp paralel for*. Hal ini bertujuan agar *master thread* membangkitkan beberapa *thread* yang dibutuhkan. *Master thread* melakukan pemijahan ke masing-masing *thread* untuk mengeksekusi program secara paralel.

Masing-masing *thread* akan melakukan iterasi pada wilayah-wilayah tengah citra dan juga mengerjakan fungsi Sobel, sehingga kode yang mengalami pemijahan akan melakukan semua fungsi secara paralel dalam *looping* maupun dalam mengerjakan fungsi Sobel. Ketika semua *thread* telah melakukan *task*-nya sampai selesai, tahap selanjutnya adalah menggabungkan kembali hasil dari eksekusi masing-masing *thread* pada *master thread.*

3.2.1 Pengembangan Pemrograman Serial Ke MPI

Pemrograman paralel dengan MPI mengharuskan *programmer* untuk mengatur aliran data antar proses secara eksplisit. *Programmer* perlu memodifikasi bagian kode serial untuk memudahkan proses pembagian data. Pada MPI, *master* dan *slave* saling berhubungan untuk bekerja secara paralel. *Master* bertanggung jawab untuk mengatur dan mendistribusikan data, sementara *slave* beroperasi pada *task* yang diterima.

Adapun *flowchart* pemrograman paralel dengan MPI pada deteksi tepi menggunakan operator Sobel seperti terlihat pada Gambar 3.6.



Gambar 3.6 *Flowchart* pemrogaman paralel MPI deteksi tepi menggunakan operator Sobel

Pada pemrograman paralel MPI deteksi tepi Sobel, sebuah program ditulis untuk dijalankan di atas semua *processor* yang ada. Proses pertama yang harus dilakukan adalah inisialisasi MPI *runtime*. Fungsi inisialisasi yang digunakan adalah *MPI_Request* untuk meminta penanganan *task*, *MPI_Init* untuk inisialisasi lingkungan MPI dan *MPI_Comm_rank* untuk menentukan *rank* yang digunakan. Kode program yang digunakan untuk langkah ini yaitu:

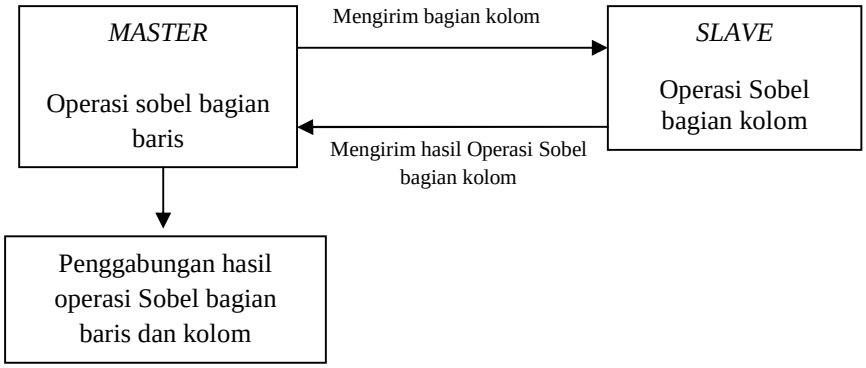
```
......
MPI_Request request;
awal=MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
......
```

Langkah selanjutnya adalah menentukan batas wilayah baris dari citra. Kode program yang digunakan untuk langkah ini yaitu:

```
.........
 int startY=(rank*grey.rows);
 if(startY==0)
{
        startY=1;
 }
int stopY=((rank+1)*grey.rows);
if(stopY>grey.rows - 1)
{
        stopY=grey.rows - 1;
}
for(int y = startY; y < stopY; y++)
......
```

startY dan *stopY* adalah batas penentuan wilayah baris pada citra. *startY* adalah inisialisasi wilayah baris pertama. Jika *startY* bernilai 0, nilai 1 di-*copy* ke *startY*, jika tidak maka dilakukan inisialisasi pada wilayah baris terakhir pada citra yaitu *stopY*. jika *stopY* lebih besar dari -1, objek *grey* dengan *method rows* di-*copy* ke stopY kemudian dikurangi 1. Hasil dari nilai-nilai penentuan batas wilayah ini digunakan untuk proses *looping* baris pada citra secara keseluruhan yaitu dari *startY* sampai dengan *stopY*.

Langkah selanjutnya adalah mengerjakan baris dan kolom untuk proses operasi Sobel agar bekerja secara paralel. Adapun model pemrogaman paralel dengan MPI pada deteksi tepi menggunakan operator Sobel seperti terlihat pada Gambar 3.7.



Gambar 3.7 Model pemrogaman paralel MPI pada deteksi tepi Sobel

Dalam tahap ini pembagian *task*-nya telah ditentukan pada saat awal. *Master* membagi task menjadi sub-*task*, kemudian mengirimnya ke *slave*. Komunikasi yang terjadi dalam kode program ini adalah komunikasi global yang terjadi pada saat *master* mengerjakan bagian baris, dan *master* mengirimkan bagian kolom ke *slave*, agar segera dilakukan proses operasi Sobel. Kode program yang digunakan untuk langkah ini yaitu:

```
{
if (rank==0)
    MPI_Isend(&x, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &request);
Else if (!rank==0)
     MPI_Irecv(&x, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &request);

    for(x = 1; x < grey.cols - 1; x++)
    {
       gx = xGradient(grey, x, y);
       gy = yGradient(grey, x, y);
       sum = abs(gx) + abs(gy);
       sum = sum > 255 ? 255:sum;
       sum = sum < 0 ? 0 : sum;
      dst.at<uchar>(y,x) = sum;
     }
}
```

Data yang dikirim ke *slave* adalah bagian kolom, kemudian dihitung (*count*) banyaknya data yang dikirim dan diterima yaitu 1 dengan tipe data *integer* (MPI_INT), *Source* pengiriman data adalah 0, *Source* penerimaan data adalah 1, nilai *Message tag* adalah 1, dan nama *output* komunikasi adalah *request*.

Setelah operasi Sobel selesai dikerjakan oleh *master* dan *slave*, *master* menyimpan citra hasil deteksi operator Sobel ke *folder* yang telah ditentukan, yaitu *folde*r MPI dan menampilkan hasil waktu eksekusi pada *command prompt*. Kode program yang digunakan untuk langkah ini yaitu:

```
.....
if (rank==0)
{
     imwrite( "E:/tigaout/Debug/MPI/output deteksi Tepi Sobel.bmp", dst);
    cout<<"waktu eksekusi adalah: "<< end-start << " detik " <<endl;
 }
.....
```

Diakhir program paralel MPI deteksi tepi Sobel, program diakhiri dengan melakukan terminasi lingkungan eksekusi MPI. Kode program yang digunakan untuk langkah ini yaitu:

```
….
MPI_Finalize ();
….
```

3.4 Pengambilan Data Waktu Eksekusi

Penelitian implementasi pemrograman paralel dalam deteksi tepi menggunakan metode operator Sobel dalam pustaka pemrograman OpenMP dan MPI menggunakan jenis metode penelitian eksperimen, dimana dengan mengambil 10 data eksperimen pada pemrograman serial dan pemrograman paralel. Kemudian kode program tersebut dijalankan untuk mendapatkan *output* citra asli, citra *grayscale*, citra deteksi tepi Sobel dan hasil waktu eksekusi.

Untuk menganalisis kinerja metode Sobel, jenis citra *input* yang digunakan untuk pengujian program deteksi tepi Sobel adalah citra warna yaitu sebuah citra digital berekstensi BMP dengan tingkat ukuran *file* yang berbeda (5,11 megabyte, 8,58 megabyte, 19 megabyte, 30,4 megabyte dan 50 megabyte). Adapun spesifikasi komputer yang digunakan dalam penelitian ini adalah seperti terlihat pada Tabel 3.1.

Tabel 3.1 Spesifikasi *hardware* yang digunakan

Peripheral	Keterangan
RAM	2,00 gigabyte DDR-3
Prosesor	*Intel(R) Core(TM)* i3 CPU @2.4 Ghz
Sistem Operasi	*Windows* 7 *Ultimate* (32 bit)
Aplikasi Pendukung	*Visual Studio Express* 2012

3.5 Pengolahan Data Waktu eksekusi

Dalam penelitian ini, pengolahan data dilakukan dengan cara menghitung rata-rata waktu eksekusi yang diperoleh dari data 10 kali percobaan terhadap setiap kode program yang dilakukan pada tahapan sebelumnya. Langkah selanjutnya adalah melakukan rasio rata-rata waktu eksekusi program serial dan rata-rata waktu eksekusi program paralel

menggunakan pustaka OpenMP dan MPI untuk memperoleh berapa banyak peningkatan kecepatan (*speedup*) yang diperoleh.

Pertama-tama *master* membagi *task* menjadi beberapa sub *task*, kemudian mengirimkannya ke *slave*. Pendistribusian *task* ke suatu *slave* tergantung dari kemajuan proses dari tiap *slave*. Bila *slave* selesai memproses suatu *task*, maka akan segera menerima *task* berikutnya. pembagian *task*-nya telah ditentukan pada saat awal.

Dalam pemrograman MPI ini tidak ada masalah penggabungan data, tetapi cukup membebani *master* dalam membagi *task*. *Master* membutuh waktu lama untuk membagi *task*, sementara itu *slave* menunggu kiriman *task*. Hal ini menyebabkan kinerja algoritma paralel MPI secara keseluruhan berkurang. Dalam komunikasi *point to point non blocking* ini, *task* dapat segera dibagi dan *slave* segera bekerja.

Master bertanggung jawab untuk mengatur dan mendistribusikan data sementara
slave beroperasi pada *task* yang diterima

sedangkan pada model *message passing* memorinya tersebar di masing-masing prosesor. Tiap prosesor memiliki alamat memorinya masing-masing, dan prosesor yang perlu mengakses memori yang bukan miliknya harus melakukan komunikasi antar prosesor. Sehingga, dalam hal ini, komunikasi dilakukan secara eksplisit, yang melibatkan waktu I/O.

pemrograman paralel dengan MPI ini lebih rumit dibandingkan dengan OpenMP, karena MPI mengharuskan *programmer* untuk mengatur aliran data antar proses secara eksplisit.

Komunikasi dan sinkronisasi antar proses dilakukan dengan pengiriman dan penerimaan *task* secara jelas dan teratur.

startY dan *stopY* adalah batas delimitasi baris pada citra. Pada *startY*, *master* melakukan penelusuran baris pada batas citra *(rank*grey.rows)* sementara pada *stopY*, *slave* melakukan penelusuran baris pada tengah citra *((rank+1)*grey.rows).* Hasil penelusuran ini akan diterima oleh *master*, untuk dilakukan iterasi baris secara keseluruhan pada citra dari *starY* sampai dengan *stopY*.

Komunikasi yang terjadi dalam kode program ini adalah komunikasi global yang terjadi pada saat *master* mengerjakan baris dan *master* mengirimkan kolom citra ke *slave*.
dan pada saat masing-masing
processor mengirimkan hasil filterisasi yang dilakukan terhadap sub matriks citra yang telah dibagikan sebelumnya kepada *root*.

Secara umum algoritme ini dibagi ke dalam 3 bagian yaitu inisialisasi awal (pembacaan matriks, ukuran baris dan ukuran kolom) oleh *root*, filterisasi sub matriks citra oleh semua *processor* dan penggabungan hasil akhir ke *root*.

Untuk mengimplementasikan desain paralelisme menggunakan algoritme dan metode Foster yang telah didefinisikan sebelumnya, maka program diimplementasikan ke dalam bahasa C menggunakan MPI. Programnya adalah sebagai berikut: