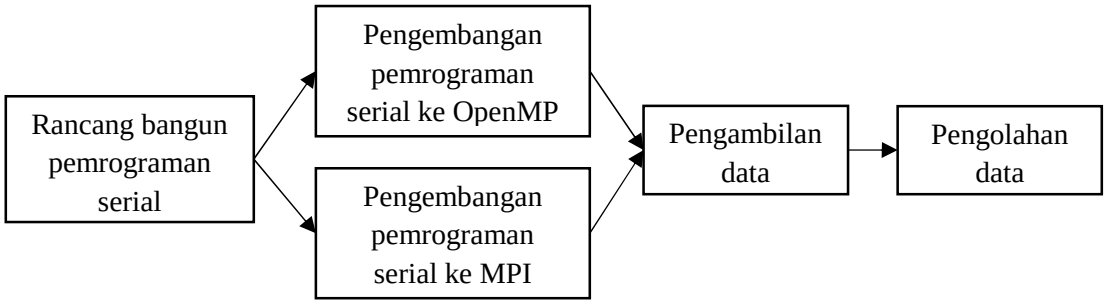
BAB III

METODE PENELITIAN

Pada penelitian implementasi pemrograman paralel pada deteksi tepi dengan menggunakan metode *Canny,* program komputerdiprogram dengan bahasa pemrograman C++. Dalam penelitian ini, metode yang digunakan untuk mengambil data adalah dengan menggunakan sebuah eksperimen dengan menjalankan program-program yang telah dibuat pada prosesor PC yang digunakan pada penelitian ini.

Untuk mendapatkan hasil dari penelitian diperlukan beberapa langkah penelitian. Langkah-langkah penelitian tersebut dapat disimpulkandalam diagram blok seperti terlihat pada Gambar 3.1.



Gambar 3.1 Diagram blok metode peneltian

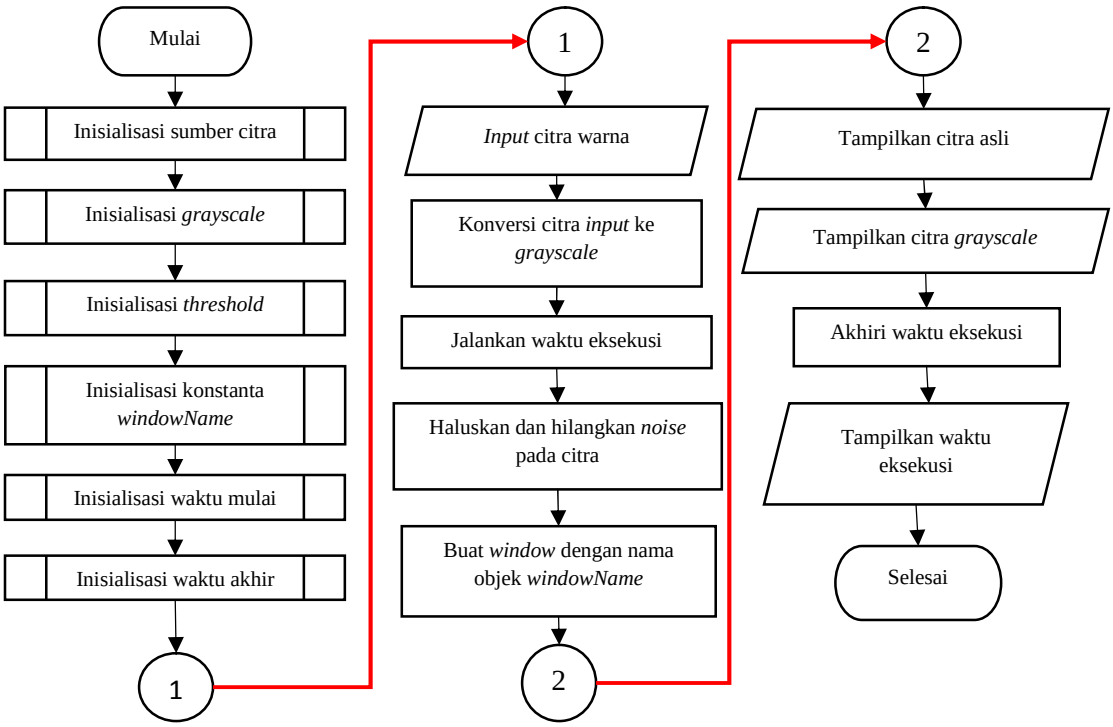
Pembuatan diagram blok digunakan untuk menampilkan langkah-langkah dalam penelitian ini. Langkah yang pertama yaitu merancang dan membangun pemrograman serial dengan menggunakan metode *Canny* dalam bahasa pemrograman C++.Untuk mempermudah pembuatan pemrograman serial, terlebih dahulu dibuat*flowchart* pemrograman serial, kemudian *flowchart* tersebut diterjemahkan dalam kode-kode program dalam bahasa pemrograman C++.

Langkah selanjutnya adalah pengembangan pemrograman serial ke pemrograman paralel. Dalam pemrograman paralel ada dua pustaka yang digunakan yaitu pustaka OpenMP dan pustaka MPI. Langkah ketiga adalah pengambilan data waktu eksekusi dengan cara menjalankan kode program yang telah dibuat baik pemrograman serial maupun pemrograman paralel sebanyak 10 kali, kemudian waktu eksekusi dicatat.

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

3.1 Rancang Bangun Pemrograman Serial

Dalam merancang dan membangun pemrograman serial pada deteksi tepi dengan menggunakan metode *Canny*, diperlukan pembuatan *flowchart* pemrograman serial. *Flowchart* yang telah dibuat kemudian diterjemahkan dalam bentuk potongan-potongan kode program dengan menggunakan bahasa pemrograman C++. *Flowchart* adalah bagan alir yang menceritakan bagaimana langkah-langkah dan prosedur-prosedur pada suatu program.*Flowchart* juga memberikan suatu kemudahan bagi seorang *programmer* untuk membuat sebuah program. Setelah *flowchart* selesai dibuat kemudian dilanjutkan dengan penulisan kode program sesuai dengan *flowchart* yang telah dibuat. Adapun *flowchart*pemrograman serial deteksi tepi menggunakan metode *Canny*seperti terlihat pada Gambar 3.2.



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

Pada pemrograman serial deteksi tepi menggunakan metode *Canny*, program dimulai dengan inisialisasi data-data yang diperlukan dalam deteksi tepi.Mulai dari inisialisasi sumber citra yang digunakan, inisialisasi *grayscale,* inisialisasi konstanta *windowName*, hingga inisialisasi waktu ekekusi. Pada inisialisasi *threshold* skala yang dipakai mulai dari 15, karena pada skala 15 program dapat dijalankan. Jika *threshold* dibawah 15, maka program yang dijalankan membutuhkan waktu yang sangat lama pada citra yang berukuran besar. Kode program inisialisasi secara berturut-turut yaitu:

```
……..
Mat src, grey;
int thresh = 15;
const char* windowName = "Contours";
clock_t start, finish;
 …….
```

Langkah selanjutnya adalah peng-*input*-an citra digital dari sebuah *file*yang akan dideteksi.*Input* dari program yang dibuat adalah *file* citra berwarna dengan jenis *file* citra *bitmap* (BMP).Fungsi *imread*adalah untuk membaca citra yang akan digunakan pada deteksi tepi *Canny.*Sintaks dasar yang digunakan pada *imread* adalah sebagai berikut:

```
……
src = imread("2.jpg");
 ……
```

Pada sintaks dasar pemrograman di atas,“src” adalah sebagai objek untuk menyimpan sementara *file* gambar, sedangkan “imread”adalah objekdari pustakaOpenCV yang berfungsi untuk membaca citra.

Citra yang telah di-*input* kemudian dikonversi ke *grayscale* untuk mendapatkan keabuan pada citra. Untuk mengubah gambar RGB ke gambar *grayscale* dapat dilakukan dengan cara mengambil piksel pada gambar kemudian warna tiap pikselakan diambil informasi mengenai 3 warna dasar yaitu merah, biru dan hijau. Ketiga warna dasar ini akan dijumlahkan kemudian dibagi tiga sehingga didapat nilai rata-rata. Nilai rata-rata inilah yang akan dipakai untuk memberikan warna pada piksel gambar sehingga warna menjadi

grayscale.Tiga warna dasar sebuah piksel akan diset menjadi nilai rata-rata.Sintaks dasar yang digunakan untuk*grayscale* adalah sebagai berikut:

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

"cvtColor " pada sintaks dasar berfungsi untuk mengkonversi gambar warna. "src" yaitu *file* gambar yang akan dikonversi, sedangkan "grey" adalah *file* hasil dari konversi, "CV_BGR2GRAY" yaitu objekdari pustakaOpenCV yang berfungsi untuk mengubah citra warna menjadi citra keabuan *(grayscale)*.

Setelah proses konversicitra ke*grayscale*selesai, maka langkah selanjutnya adalah menjalankan waktu eksekusi. Sintaks dasar yang digunakan untuk menjalankan waku eksekusiadalah sebagai berikut:

```
…..
start=clock();
…..
```

Setelah cacah waktu berjalan, langkah selanjutnya adalah citra *grayscale* dihaluskan dan dihilangkan *noise*-nya dengan menggunakan fungsi *gaussian blur*. Sintaks dasar yang digunakan pada *gaussian blur* adalah sebagai berikut:

```
……
cv::GaussianBlur(grey,grey,Size(3,3),0);
……
```

"cv::GaussianBlur" adalah fungsi dari OpenCV yang bertugas untuk menghaluskan dan menghilangkan *noise. grey* yang pertama adalah citra yang akan diproses pada *gaussian blur*, sedangkan *grey* yang kedua adalah hasil dari *blurring.* "size(3,3)" adalah ukuran gambar dengan tinggi 3 dan lebar 3, dan "0" adalah nilai standar deviasi dari *gaussian* baik dalam arah sumbu Y maupun sumbu X.

Langkah selanjutnya adalah pembuatan *window* untuk menampilkan citra dengan nama fungsi sesuai dengan citra yang akan ditampilkan. Sintaks dasar yang digunakan pada pembuatan *window* adalah sebagai berikut:

```
……
namedWindow(windowName);
……
```

"namedWindow" berfungsi untuk membuat tampilan *window highgui,* sedangkan "windowName" adalah nama jendela yang digunakan sebagai induk citra yang ditampilkan.

Langkah selanjutnya adalah menampilkan citra asli dan citra*grayscale*pada masing-masing *window* yang telah dibuat. Sintaks dasar yang digunakan adalah sebagai berikut:

```
……
namedWindow("Original");
imshow("Original",src);
namedWindow("grey");
imshow("grey",grey);
……
```

Objek “namedWindow” berfungsi untuk membuat tampilan *window highgui*, sedangkan “imshow” adalah perintah untuk menampilkan citra.

Langkah selanjutnya adalah membuat *trackbar* pada *window contours*dengan interval 0-255.*Trackbar* ini berfungsi sebagai pengatur tingkat kecerahan pada citra hasil deteksi tepi *Canny.* Sintaks dasar yang digunakan adalah sebagai berikut:

```
……
cv::createTrackbar("Thresholding",windowName,&thresh,255,detectContours);
…….
```

“cv::createTrackbar” adalah objek dari pustaka OpenCV untuk membuat *trackbar* pada*window*yaitu *thresholding.* “windowName” adalah objek dari nama *window threshold,* “&thresh” adalah nilai awal dari *threshold,* yaitu sebesar 255, “255” adalah nilai maksimum *range*dari *threshold,* sedangkan “detectContours” adalahfungsi yang dipanggil.

Langkah selanjutnya adalah memanggil fungsi *detectContours.*Pada fungsi tersebut terdapat proses-proses deteksi tepi *Canny.* Adapun pemanggilan fungsi *detectContours*adalah sebagai berikut:

```
…….
detectContours(0,0);
…….
```

“(0,0)” adalah nilai argumen untuk menghubungkan dengan variabel pada fungsi detectcountours(). Pada fungsi *detectContours* terjadi beberapa proses diantaranya mendeteksi tepi dengan metode *Canny,* menemukan *contours*dan menggambarkannya.

Fungsi *detectContours*dipanggil oleh fungsi utama. Sintaks dari *detectContours*adalah sebagai berikut:

```
……
void detectContours(int,void*)
 ……
```

“void detectContours” adalah fungsi tanpa tipe data dengan nama fungsi *detectContours,* sedangkan “int,void*” adalah tipe data argumen untuk menghubungkan dengan variabel pada fungsi utama yaitu detectContours(0,0). Selanjutnya pada *detectContours* terdapat dua variabel dengan tipe data Mat. Sintaks dari variabel tersebut adalah:

```
…….
Mat canny_output,drawing;
 ……
```

“Mat” merupakan kelas 2D atau berdimensi banyak yang berbentuk *array* yang dapat menyimpan matriks, gambar, histogram, penggambaran fitur dan lain-lain,sedangkan “canny_output, drawing” merupakan variabel yang bertipe dataMat.

Langkah selanjutnya yang terdapat pada detectContours() adalah penyimpanan sementara detectContours()yang disimpan sebagai *vector point.* Adapun sintaks untuk penyimpanan hasil sementara detectContours() adalah sebagai berikut:

```
……..
vector<vector<Point>> contours;
vector<Vec4i>heirachy;
 ……..
```

Selain penyimpanan sementara detectContours() terdapat juga objek OpenCV untuk melakukan deteksi tepi dengan menggunakan metode *canny.* Adapun sintaks untuk mendeteksi dengan metode *Canny* adalah sebagai berikut:

```
……...
cv::Canny(grey,canny_output,thresh,2*thresh,3);
 ……...
```

“cv::Canny” adalah objek dari OpenCV yang berfungsi untuk menemukan tepian gambar dengan menggunakan algoritma *Canny,* “grey” adalah citra input yang akan diproses dengan menggunakan metode *Canny,* “canny_output” adalah hasil dari deteksi tepi dengan metode *Canny,* sedangkan “thresh” adalah batas ambang pertama untuk prosedur histeresis dan “2*thresh” adalah batas ambang kedua untuk prosedur histeresis.

Pada deteksi *contours* terdapat juga perintah untuk menampilkan hasil citra deteksi tepi dengan menggunakan metode *Canny* dan hasil dari *contours.* Adapun sintaks untuk menampilkan deteksi tepi *Canny* dan *contours* adalah sebagai berikut:

```
……
NamedWindow("Canny");
imshow("Canny",canny_output);
```

```
imshow(windowName,drawing);
……..
```

Objek “namedWindow” berfungsi untuk membuat tampilan *window highgui*, sedangkan “imshow” adalah perintah untuk menampilkan citra

Perintah selanjutnya pada deteksi *contours* adalah mencari tingkat kecerahan hasil deteksi tepi pada *contours* dengan mengubah nilai pada *trackbar*. Adapun sintaks untuk mencari kecerahan pada hasil deteksi tepi *Canny* adalah sebagai berikut:

```
…….
cv::findContours(canny_output,contours,heirachy,CV_RETR_TREE,CV_CHAIN_APPROX_
SIMPLE,Point(0,0));
…….
```

“cv::findContours” adalah objek dari OpenCV untuk mencari *contours,* “canny_output” adalah citra input yang akan dicari kecerahannya, sedangkan “contours” merupakan hasil dari pencerahan. “heirachy” adalah opsional keluaran vector yang berisi tentang informasi topologi gambar, karena pada *heirachy* memiliki banyak elemen sebagai jumlah *contours.*“CV_RETR_TREE” berfungsi untuk menemukan kembali semua *contours* dan memperbaiki susunan *nested contours*secara penuh. “CV_CHAIN_APPROX_SIMPLE” adalah pengkompresan segmen horizontal, vertikal, dan diagonal pada titik akhir *contours*. “point” adalah titik 2D pada kelas *template*, sedangkan “0,0” adalah nilai sementara dari *contours* baik secara horizontal dan vertikal.

Selanjutnya perintah yang terdapat pada deteksi *contours* adalah membuat latar belakang citra yang berwarna hitam. Adapun sintaks untuk pembuatan latar belakang citra adalah sebagai berikut:

```
…….
drawing = Mat::zeros(canny_output.size(),CV_8UC1);
…….
```

“drawing” adalah variabel untuk menyimpan nilai konstanta dari sebuah objek *zeros*, yaitu untuk membuat warna hitam, “canny_output.size()” adalah nilai piksel dari citra yang sudah dideteksi tepi dengan metode *size*, “CV_8UC1” adalah nilai-nilai piksel dari gambar hitam putih

Perintah selanjutnya yang terdapat pada deteksi *contours* adalah menggambar *contours*. Adapun sintaks untuk menggambar *contours* adalah sebagai berikut:

```
……..
for(int i=0;i<contours.size();i++)
{
cv::drawContours(drawing,contours,i,Scalar(255),1,8,heirachy,0,Point());
}
 ……..
```

“for” adalah pengulangan *increment* mulai dari 0 sampaikurang dari ukuran *contours*. “cv::drawContours” adalah objek dari OpenCV untuk menggambar *contours*, “drawing” adalah *input* berupa gambar yang akan digambar pada *contours*, “contours” adalah *input* berupa garis tepi, “i” adalah nilai *index*dari *contours*. “scalar” adalah vektor 4 elemen, “255” adalah jumlah maksimum dari kecerahan citra, “1” adalah ketebalan garis pada *contours*, “8” adalah tipe garis yang terdapat pada*contours*,“heirachy” adalah fungsi untuk menarik *contours* tertentu,“0” adalah tingkat maksimal untuk memanggil *contours*, nilai 0 berarti *contours* yang dipanggil hanya pada*contours* tertentu, sedangkan “point” adalah adalah titik 2D pada kelas *template*.

Langkah selanjutnya adalah mengatur *trackbar* pada *window contour* untuk menentukan tingkat kecerahan pada citra *output* deteksi tepi *Canny*. Namun apabila pada citra tersebut tingkat kecerahan sudah cukup cerah sesuai kebutuhan, maka langkah selanjutnya adalah menyimpan citra pada folder yang telah ditentukan. Adapun citra yang disimpan adalah citra asli, citra *grayscale*, dan citra deteksi tepi *Canny*. Sintaks dasar yang digunakan pada penyimpanan citra adalah sebagai berikut:

```
……..
imwrite("canny.bmp",canny_output);
imwrite("grey.bmp",grey);
imwrite("original.bmp",src);
 …….
```

“imwrite” adalah perintah untuk menyimpan hasil dari citra yang telah diproses.

Langkah selanjutnya adalah mengakhiri waktu eksekusi. Adapun sintaksyang digunakan adalah sebagai berikut:

```
……..
finish=clock();
 ……..
```

Setelah waktu eksekusi diakhiri, maka waktu eksekusi ditampilkan pada *command prompt* dengan cara mengurangi waktu akhir dengan waktu awal. Adapun sintaks yang digunakan untuk menampilkan waktu eksekusi adalah sebagai berikut:

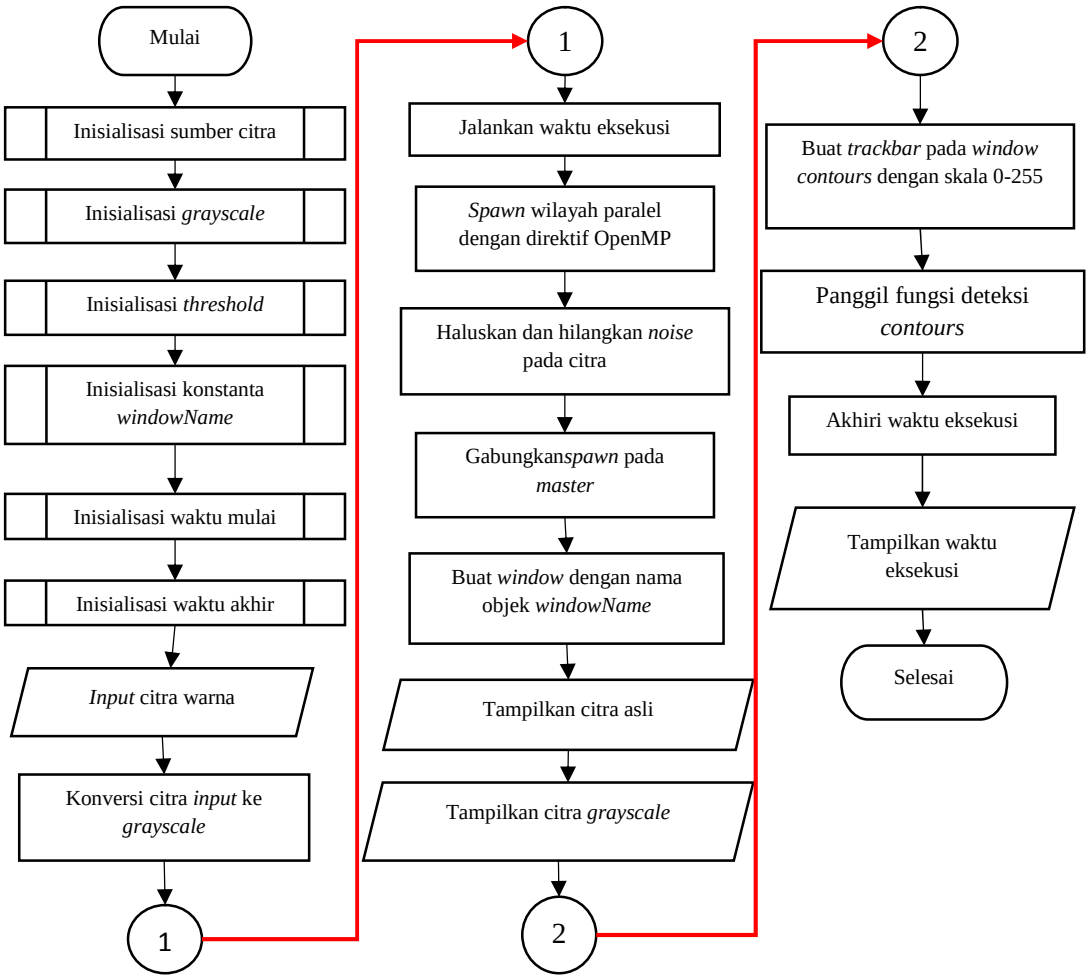
```
……..
 cout<<"waktu eksekusi="<<float(finish-start)/CLOCKS_PER_SEC<<" DETIK"<<endl;
 …….
```

Setelah waktu eksekusi ditampilkan maka proses pada deteksi tepi dengan menggunakan metode *Canny*telahselesai dan hasil deteksi tepi dapat dilihat pada *folder* yang telah ditentukan.

3.2 Pengembangan Pemrograman Serial Ke Pemrograman Paralel

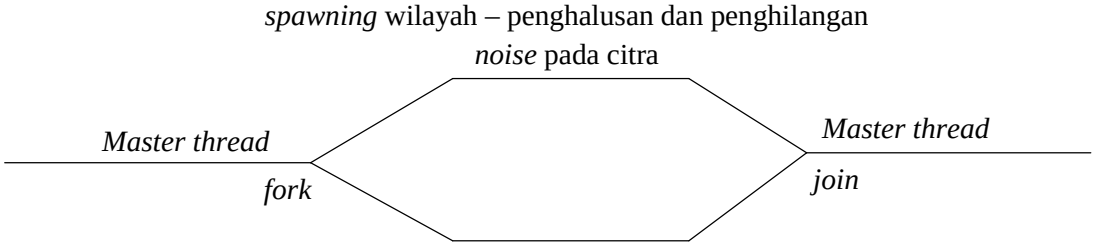
3.2.1Pengembangan Pemrograman Serial Ke OpenMP

Setelah rancang bangun pemrograman serial telah selesai, selanjutnya dikembangkan pemrograman serial tersebut ke pemrograman paralel dengan menggunakan pustaka OpenMP. Pengembangan pemrograman serial ke OpenMP dilakukan dengan cara merancang *flowchart*, kemudian *flowchart* yang telah jadi diterjemahkan dalam bentuk program dalam bahasa C++. *Flowchart* sangatlah penting sebagai tahap awal dari perancangan sebuah algoritma, karena *flowchart* menggambarkan langkah-langkah program yang akan dibuat. Setelah *flowchart* dirancang, selanjutnya adalah penulisan kode program paralel dengan menggunakan pustaka OpenMP ke dalam bahasa C++.Adapun *flowchart*pengembangan pemrograman serial deteksi tepi menggunakan metode *Canny*ke pemrograman paralel dengan menggunakan pustaka OpenMP seperti terlihat pada Gambar 3.3.



Gambar 3.3 *Flowchart* pemrograman OpenMP deteksi tepi menggunakan metode *Canny*

Model dari pemrograman OpenMP untuk *spawning* wilayah agar masing-masing *thread* dapat menghaluskan dan menghilangkan*noise* pada citra dapat dilihat pada gambar 3.4



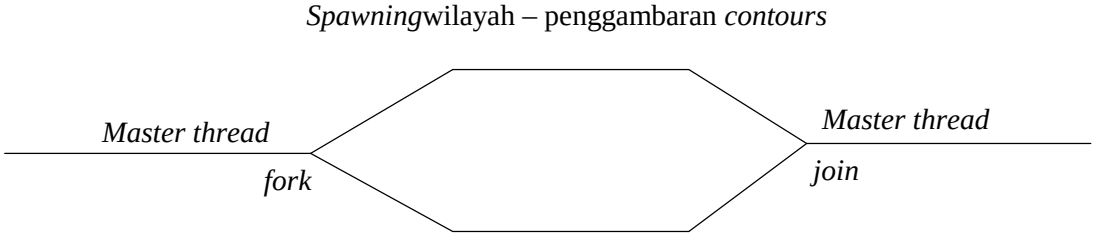
Gambar 3.4 Model pemrograman OpenMP untuk penghalusan dan penghilangan *noise* pada citra

Paralelisasi dengan menggunakan direktif *parallel* untuk membangkitkan *thread* agar konvolusi yang terjadi pada penghalusan dan penghilangan *noise* pada citra dapat bekerja secara bersama. Piksel pada citra dibagi sesuai dengan banyaknya jumlah *thread,* matriks kolom dikerjakan oleh*thread*nol dan matriks baris dikerjakan oleh *thread* satu.Ketika semua *thread* telah melakukan *task*-nya sampai selesai, tahap selanjutnya adalah menggabungkan kembali hasil dari eksekusi masing-masing *thread*pada *master thread.*Sintaks dasar dari paralelisasi pada penghalusan dan penghilangan*noise* adalah sebagai berikut:

```
……
#pragma omp parallel
{
 cv::GaussianBlur(grey,grey,Size(3,3),0);
}
……
```

"#pragma omp" adalah direktif *compiler,* sedangkan "parallel" adalah nama direktif.

Langkah selanjutnya adalah memparelisasikanproses penggambaran *contours*yang terdapat pada fungsi detectContours( ).Pada fungsi detectContours( )terjadi beberapa proses diantaranya mendeteksi tepi dengan menggunakan metode *Canny,* menemukan *contours,* dan menggambarkannya.Adapun model pemrograman OpenMP didalam fungsi detectContours( )seperti terlihat pada Gambar 3.5.



Gambar 3.5 Model pemrograman paralel OpenMP untuk pemanggilan fungsi *detectContours*

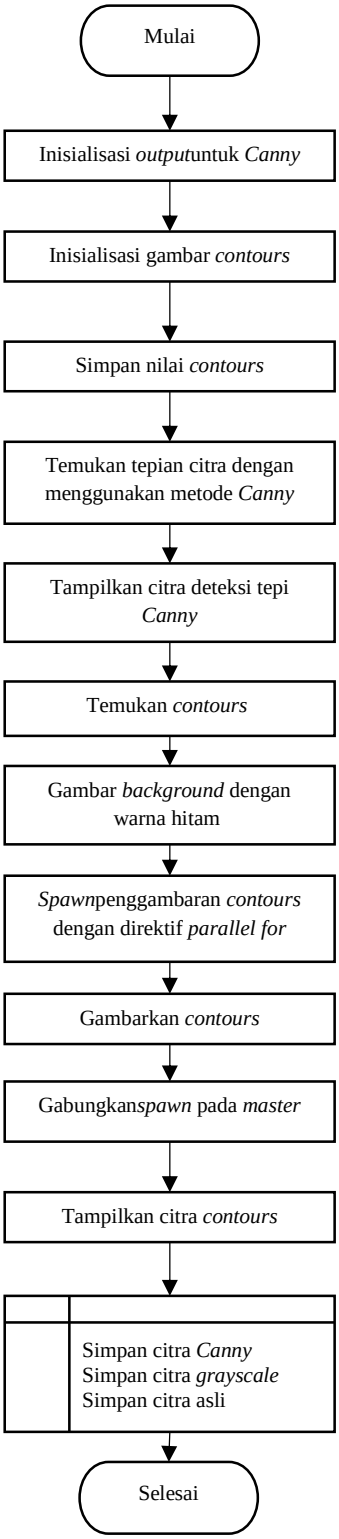
Paralelisasi pada penggambaran *contours* ini bertujuan agar *master thread* membangkitkan beberapa *thread* yang dibutuhkan. *Master thread* melakukan *spawning* di masing-masing *thread* untuk mengeksekusi pemrograman secara paralel. Masing-masing *thread* akan melakukan proses-proses yang terjadi pada *iterasi* penggambaran *contours,* sehingga kode program yang mengalami *spawning* akan melakukan semua fungsi secara paralel dalam iterasi

penggambaran *contours*. Ketika semua *thread* telah melakukan *task*-nya sampai selesai, tahap selanjutnya adalah menggabungkan kembali hasil dari eksekusi masing-masing *thread* pada *master thread.*

Paralelisasi yang dilakukan didalam fungsidetectContours( )bertujuan untuk mempercepat proses-proses yang terjadi pada fungsi detectContours( )tersebut. Sintaks paralelisasi didalam fungsi detectContours( ) untukpenggambaran *contours* adalah sebagai berikut:

```
.........
#pragma omp parallel for num_threads(2)
For(int i=0;i<contours.size();i++)
   {
      cv::drawContours(drawing,contours,i,Scalar(255),1,8,heirachy,0,Point());
   }
.........
```

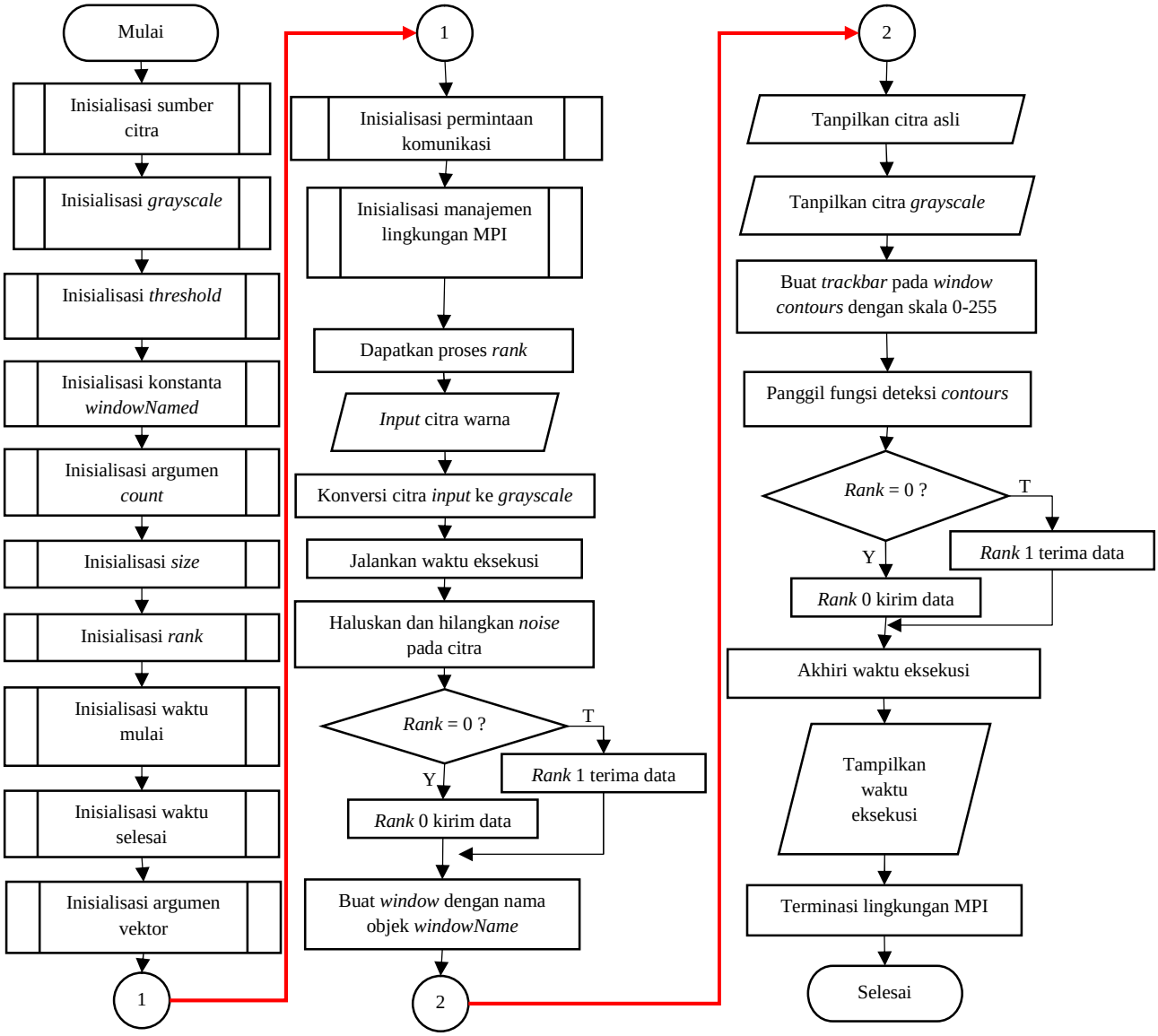
Flowchart didalam fungsi detectcontours( )seperti terlihat pada Gambar 3.6.



Gambar 3.6 *Flowchart* pemrograman paralel OpenMP di dalam fungsi detectContours( )untuk penggambaran *contours*

3.2.2 Pengembangan Pemrograman Serial Ke MPI

Pengembangan pemrograman serial ke MPI pada deteksi tepi dengan menggunakan metode *Canny*, ada beberapa hal yang harus diperhatikan oleh seorang *programmer*. Pada MPI,*master* dan *slave* saling berhubungan untuk bisa bekerja secara paralel seperti yang diharapkan.*Master* bertanggung jawab untuk mengatur dan mendistribusikan data,sedangkan*slave* beroperasi pada *task* yang diterima. Adapun *flowchart* pemrograman paralel dengan menggunakan pustaka MPI pada deteksi tepi dengan menggunakan metode *Canny* seperti terlihat pada gambar 3.7.



Gambar 3.7*flowchart* pemrograman paralel MPI deteksi tepi menggunakan metode *Canny*

Dilihat dari gambar *flowchart*di atas, pemrograman paralel dengan pustaka MPI dimulai dengan inisialisasi data-data yang diperlukan dalam pemrograman paralel. Sintaks dasar dari inisialisasi pemrograman paralel adalah sebagai berikut:

```
.....
int argc, size, rank;
double start, end;
char **argv;
MPI_Request request;
MPI_Init(&argc, &argv);
.....
```

“argc” adalah variabel untuk menghitung jumlah argumen, dan“size” adalah variabel untuk mencari banyak prosesor, sedangkan “rank” adalah variabel komunikator. “double” adalah tipe data untuk waktu, dan “start” adalah variabel untuk waktu mulai, sedangkan “end” adalah variabel untuk penghentian waktu. “char” adalah tipe data untuk argumen vektor, sedangkan “argv” adalah variabel untuk menentukan arah. “MPI_Request request” adalah fungsi untuk meminta komunikasi, sedangkan “MPI_Init” adalah fungsi inisialisasi untuk manajemen lingkungan MPI, dan “&argc” adalah variabel untuk menunjukkan jumlah argumen, sedangkan “&argv” adalah variabel yang menunjukkan arah vektor.

Proses yang dilakukan selanjutnya adalah memanggil fungsi*rank*pada komunikator,sintaks dasarnya adalah sebagai berikut:

```
.....
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
.....
```

“MPI_Comm_rank” adalah fungsi dari MPI yangberfungsi untuk mengembalikan *rank* dari suatu proses, sedangkan “MPI_COMM_WORLD” adalah komunikatoryang mendefinisikan jangkauan operasi komunikasi, dan “&rank” adalah inisialisasi dari MPI_COMM_rank.

Langkah selanjutnya adalah memulai waktu cacah pada pemrograman paralel dengan pustaka MPI, sintaks dasarnya adalah sebagai berikut:

```
.....
start=MPI_Wtime();
.....
```

Langkah selanjutnya adalah melakukan paralelisasi pada fungsi GaussianBlur( ),karena pada fungsi ini tejadi konvolusi matriks citra yang membutuhkan waktu cukup lama. Proses komunikasi paralel terjadi jika *rank* 0 mengirim data ke *rank* 1, dan *rank* 1 menerima data dari

rank 0. Sintaks dasar untuk pemrograman MPI pada fungsi GaussianBlur( )adalah sebagai berikut:

```
……
cv::GaussianBlur(grey,grey,Size(3,3),0);
if(rank==0)
{
MPI_Isend(&getGaussianKernel, 3, MPI_INT, 0, 1, MPI_COMM_WORLD, &request);
}
Else if (rank==1)
{
MPI_Irecv(&getGaussianKernel, 3, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
}
```

"if(rank==0)" adalah *rank* 0mengirim data, sedangkan "MPI_Isend" adalah sebuah operasi yang digunakan untuk mengirim data secara *non-blocking* dan bersifat segera (*immediate*). "&getGaussianKernel" adalah data yang dikirim ke *rank* 1. "3" adalah jumlah *buffer* data yang dikirim. "MPI_INT" adalah tipe data dari *buffer* yang dikirim. "0" adalah tujuan *rank* yang dikirim."1" adalah jumlah pesan *tag* yang dikirim."MPI_COMM_WORLD" adalah komunikator yang digunakan untuk mengirim data."&request" adalah *output* dari komunikasi. "Else if (rank==1)" adalah *rank* 1 menerima data."MPI_Irecv" adalah sebuah operasi yang digunakan untuk menerima data secara *non-blocking* dan bersifat segera (*immediate*)."&getGaussianKernel" adalah data yang diterima."3" adalah jumlah *buffer* data yang diterima. "MPI_INT" adalah tipe data dari *buffer* yang diterima."1" adalah tujuan *rank* yang diterima."0" adalah jumlah pesan *tag* yangditerima."MPI_COMM_WORLD" adalah komunikator yang digunakan untuk menerima data. "&request" merupakan *output* dari komunikasi.

Jika dilihat dari sintaks program di atas, apabila *rank* 0, maka *rank* melakukan pengiriman data yaitu berupa data *gaussian,*dan *rank* 1 melakukan penerimaan data. Kedua *rank* tersebutmasing-masing menginisialisasi *buf* yang dikirim dan diterima yaitu data *gaussian.*

Langkah selanjutnya adalah memparalelkan fungsi detectContours( ).Didalam fungsi detectConturs( )terdapat proses-proses deteksi tepi *Canny*. Sintaks dasarnya adalah sebagai berikut:

```
.....
{
detectContours(0,0);
if(rank==0)
{
MPI_Isend(&detectContours, 3, MPI_INT, 0, 1, MPI_COMM_WORLD, &request);
}
else
{
MPI_Irecv(&detectContours, 3, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
}
.....
```

Dilihat dari sintaks program di atas, jika *rank* 0 maka *rank* melakukan pengiriman data yaitu berupa data *detectcontors*,dan *rank* 1 melakukan penerimaan data *detectcontors*. Setelah detectContours( )diparalel,selanjutnya dihentikan waktu cacah dari pemrograman paralel dengan menggunakan pustaka MPI. Sintaks dasarnya adalah sebagai berikut:

```
.....
end=MPI_Wtime();
.....
```

Setelah waktu cacah dihentikan, langkah selanjutnya menampilkan waktu cacah dari pemrograman paralel dengan menggunakan pustaka MPI. Sintaks dasarnya adalah sebagai berikut:

```
…….
cout<<"Waktu Eksekusi Parallel : "<<end-start<<" Detik"<<endl;
…….
```

Langkah selanjutnya adalah menghentikan proses pemrograman paralel, sintaks dasarnya adala sebagai berikut:

```
.....
MPI_Finalize();
.....
```

Setelah pemrograman paralel dihentikan, maka proses paralelisasi dengan menggunakan pustaka MPI pada deteksi tepi dengan menggunakan metode *canny* sudah selesai.

3.3 Pengambilan Data

Pengambilan data dalam penelitian implementasi pemrograman paralel dalam deteksi tepi menggunakan metode *Canny* dalam pustaka pemrograman OpenMP dan MPI

menggunakan jenis metode penelitian eksperimen, dimana dengan mengambil 10 data eksperimen pada pemrograman serial dan pemrograman paralel.Kode program tersebut dijalankan untuk mendapatkan *output* citra asli,*output* citra *grayscale, output window contours, output* citra deteksi tepi *Canny* dan hasil waktu eksekusi.

Pada penelitian eksperimen ini menggunakan PCkomputer sebagai salah satu alat ukur untuk mendapatkan data dari masing-masing program.Spesifikasi alat ukut yang di gunakan dalam penelitian eksperimen ini dapat dilihat pada Tabel 3.1.

Tabel 3.1. Spesifikasi alat ukur.

No	Piranti	Kapasitas dan Jenis
1	Prosesor	AMD E1-2100 APU, 1.00 GHz, 350 GB
2	Memori	SATA 2,00 GB
3	*Windows OS*	*Windows* 8.0 pro
4	Aplikasi Pendukung	*Visual Studio Express* 2012

Penelitian eksperimen ini menggunakan citra sebagaisalah satu objek dalam deteksi tepi dengan menggunakan metode *Canny*.Adapun spesifikasi citra yang dipakai dalam penelitian ini dapat dilihat pada Tabel 3.2.

Tabel 3.2 Spesifikasi citra

No	Nama Citra	Ukuran Citra
1	Citra_1	12,0 MB dan 2580x1636 piksel
2	Citra_2	24,3 MB dan 3570x2381 piksel
3	Citra_3	38,2 MB dan 3872x2852 piksel
4	Citra_4	44,4 MB dan 4678x3321 piksel
5	Citra_5	51,2 MB dan 5184x3456 piksel

Citra input warna yang nantinya dikonversi ke *grayscale* kemudian diproses tepiannya dengan menggunakan metode *Canny*, baik menggunakan pemrograman serial maupun dengan pemrograman paralel dengan menggunakan pustaka OpenMP maupun MPI.

3.4 Pengolahan Data

Dalam penelitian ini, pengolahan data dilakukan dengan carastatistik, dalam ilmu statistik ada beberapa teknik yang sering digunakan, namun dalam penelitian ini teknik yang digunakan adalah teknik perata-rataan, yaitu menghitung nilai rata-rata waktu eksekusi yang diperoleh dari data 10 kali percobaan pada masing-masing program dengan banyak pengambilan data pada pemrograman serial 10 data, pemrograman paralel OpenMP 10 data dan pemrograman MPI 10 data, sehingga banyak data yang diambil dalam penelitian ini adalah sebanyak 30 data waktu eksekusi pada satu citra *input*, proses ini dilakukan pada tahapan sebelumnya. Langkah selanjutnya adalah melakukan rasio rata-rata waktu eksekusi program serial terhadap rata-rata waktu eksekusi program paralel menggunakan pustaka OpenMP maupun MPI, gunanya untuk memperoleh seberapa besar peningkatan kecepatan (*speedup*) yang diperoleh.