병렬프로그래밍실습

SOBEL EDGE DETECTION

2011270314

컴퓨터정보학과

서인석

개요

- 목적

  타겟 프로그램을 선정하여 해당 프로그램을 병렬 처리를 시도한다.

- 타겟 프로그램

  Sobel edge detection

- 실험환경

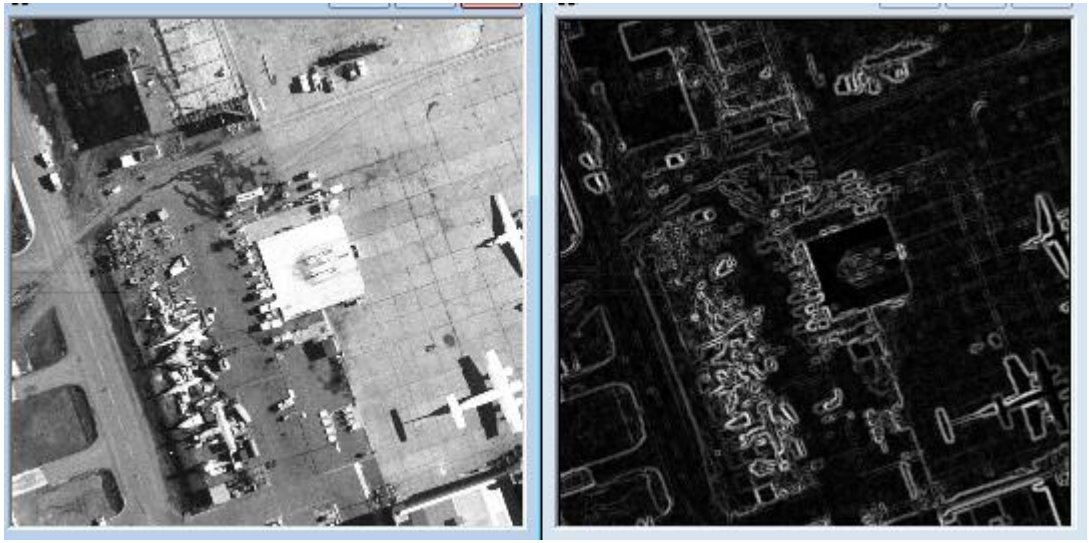
  AMD Phenom(tn) II X4 960T 3.00GHz (쿼드코어)

  NVIDIA GeForce GTX 1050

시스템 소개

- Sobel edge?

일반적으로 3x3크기의 행렬(MASK)을 사용하여 x축과 y축 별로 연산을 하고, 중심을 기준으로 각 방향의 앞뒤의 값을 비교하여 변화량을 이용해 윤곽선을 검출하는 알고리즘



성능 개선율

- 성능 개선율 계산

순차처리의 경과시간과 병렬처리의 경과시간을 기록하여 성능 개선율을 계산한다

performance improvement calculation

성능 개선율 계산

- 속도의 향상률

$$\frac{t_1 - t_2}{t_2}$$

- 예: 10분 걸리던 것이 5분 걸리면 처리속도가 2배가 된 것

  즉 속도가 100% 향상된 것 → 성능 개선율 = 100%

$$\frac{10 - 5}{5} \times 100 = 100\%$$

예상 병렬처리 부분

```
//X,Y 컨볼루젼
for (int i = 1; i < HEIGHT - 1; i++) {
    for (int j = 1; j < WIDTH - 1; j++) {
        Xval = 0;
        Yval = 0;

        for (mr = 0; mr < MASK_N; mr++) {
            for (mc = 0; mc < MASK_N; mc++) {
                Xval += (MaskSobelX[mr][mc] * InImg[(i + mr - 1) * WIDTH + (j + mc - 1)]);
                Yval += (MaskSobelY[mr][mc] * InImg[(i + mr - 1) * WIDTH + (j + mc - 1)]);
            }
        }

        ConvX[i*WIDTH + j] = Xval;
        ConvY[i*WIDTH + j] = Yval;
    }
}
```

```
 //절대값의 합계산
 for (int i = 1; i < HEIGHT - 1; i++) {
     for (int j = 1; j < WIDTH - 1; j++) {

         temp1 = ConvX[i*WIDTH + j];
         temp2 = ConvY[i*WIDTH + j];

         if (temp1 < 0)
             temp1 = -temp1;
         if (temp2 < 0)
             temp2 = -temp2;

         plmgSobel[i*WIDTH + j] = temp1 + temp2;
     }
 }
```

```
//최대값,최소값 탐색
for (int i = 1; i < HEIGHT - 1; i++) {
    for (int j = 1; j < WIDTH - 1; j++) {

        if (plmgSobel[i*WIDTH + j]<min)
            min = plmgSobel[i*WIDTH + j];
        if (plmgSobel[i*WIDTH + j]>max)
            max = plmgSobel[i*WIDTH + j];

    }
}
```

```
//값 변환
for (int i = 1; i<HEIGHT - 1; i++)
    for (int j = 1; j<WIDTH - 1; j++)
        OrgImg[i*WIDTH + j] = (unsigned char)(temp1 * plmgSobel[i*WIDTH + j] + temp2);
```

병렬처리(PTHREAD)

```
for (i = 0; i < THREAD_N; i++)
    pthread_create(&CV[i], NULL, &Conv, (void*)i);

for (i = 0; i < THREAD_N; i++)
    pthread_join(CV[i], NULL);


void *Conv(void *arg) {
    int i, j;
    int index = (int)arg * NUM;
    int th_Xval, th_Yval;
    int th_mr, th_mc;

    for (i = index + 1; (i <= index + NUM) && (i<HEIGHT - 1); i++) {
        for (j = 1; j < WIDTH - 1; j++) {

            th_Xval = 0;
            th_Yval = 0;

            for (th_mr = 0; th_mr < MASK_N; th_mr++) {
                for (th_mc = 0; th_mc < MASK_N; th_mc++) {
                    th_Xval += (MaskSobelX[th_mr][th_mc] * InImg[(i + th_mr - 1) * WIDTH + (j + th_mc - 1)]);
                    th_Yval += (MaskSobelY[th_mr][th_mc] * InImg[(i + th_mr - 1) * WIDTH + (j + th_mc - 1)]);
                }
            }

            ConvX[i*WIDTH + j] = th_Xval;
            ConvY[i*WIDTH + j] = th_Yval;
        }
    }

    return NULL;
}
```

병렬처리(PTHREAD)

```
for (i = 0; i < THREAD_N; i++)
    pthread_create(&DE[i], NULL, &Detect_edge, (void*)i);

for (i = 0; i < THREAD_N; i++)
    pthread_join(DE[i], NULL);

void *Detect_edge(void *arg) {
    int i, j;
    int index = (int)arg* NUM;

    for (i = index + 1; (i <= index + NUM) && (i<HEIGHT - 1); i++) {
        for (j = 1; j < WIDTH - 1; j++) {

            if (ConvX[i*WIDTH + j] < 0)
                ConvX[i*WIDTH + j] = -ConvX[i*WIDTH + j];
            if (ConvY[i*WIDTH + j] < 0)
                ConvY[i*WIDTH + j] = -ConvY[i*WIDTH + j];

            plmgSobel[i*WIDTH + j] = ConvX[i*WIDTH + j] + ConvY[i*WIDTH + j];

            if (plmgSobel[i*WIDTH + j] < min)
                min = plmgSobel[i*WIDTH + j];
            if (plmgSobel[i*WIDTH + j] > max)
                max = plmgSobel[i*WIDTH + j];
        }
    }

    temp1 = (float)(255.0 / (max - min));
    temp2 = (float)(-255.0*min / (max - min));

    for (i = index + 1; (i <= index + NUM) && (i<HEIGHT - 1); i++)
        for (j = 1; j < WIDTH - 1; j++)
            Orglmg[i*WIDTH + j] = (unsigned char)(temp1*plmgSobel[i*WIDTH + j] + temp2);

    return NULL;
}
```

병렬처리(CUDA 256x256)

```
__global__ void Sobel_Conv(unsigned char *d_InImg, int *d_ConvX, int *d_ConvY, int width, int height, int mr, int mc, int size) {
    int outputX = 0, outputY = 0;
    int col = threadIdx.x + blockIdx.x * blockDim.x;
    int row = threadIdx.y + blockIdx.y * blockDim.y;
    int i, j;

    int MaskSobelX[3][3] = { { -1,0,1 },
    { -2,0,2 },
    { -1,0,1 } };

    int MaskSobelY[3][3] = { { 1,2,1 },
    { 0,0,0 },
    { -1,-2,-1 } };

    if ((0 < row && row < height -1) && (0 < col && col < width-1)) {
        for (i = 0; i < mr; i++) {
            for (j = 0; j < mc; j++) {
                outputX += MaskSobelX[i][j] * d_InImg[(row + i - 1) * width + (col + j - 1)];
                outputY += MaskSobelY[i][j] * d_InImg[(row + i - 1) * width + (col + j - 1)];
            }
        }

        d_ConvX[row*width + col] = outputX;
        d_ConvY[row*width + col] = outputY;
    }

    else {
        d_ConvX[row*width + col] = 0;
        d_ConvY[row*width + col] = 0;
    }
}
```

병렬처리(CUDA 256x256)

```
__global__ void Detect_Edge(unsigned char *d_OrgImg, int *d_ConvX, int *d_ConvY,int *d_pImgSobel ,int width, int height,int *d_min,int *d_max) {

    int col = threadIdx.x + blockIdx.x * blockDim.x;
    int row = threadIdx.y + blockIdx.y * blockDim.y;
    int temp1, temp2;

    if ((row != 1 && row != height - 1) && (col != 1 && col != width - 1)) {
        if (d_ConvX[row*width + col] < 0)
            d_ConvX[row*width + col] = -d_ConvX[row*width + col];
        if (d_ConvY[row*width + col] < 0)
            d_ConvY[row*width + col] = -d_ConvY[row*width + col];

        d_pImgSobel[row*width + col] = d_ConvX[row*width + col] + d_ConvY[row*width + col];

        if (d_pImgSobel[row*width + col] < *d_min)
            *d_min = d_pImgSobel[row*width + col];
        if (d_pImgSobel[row*width + col] > *d_max)
            *d_max = d_pImgSobel[row*width + col];
    }

    __syncthreads();


    temp1 = (float)(255.0 / (*d_max - *d_min));
    temp2 = (float)(-255.0**d_min / (*d_max - *d_min));

    if ((row != 1 && row != height - 1) && (col != 1 && col != width - 1))
            d_OrgImg[row*width + col] = (unsigned char)(temp1*d_pImgSobel[row*width + col] + temp2);

}
```

256x256 이미지(PTHREAD)

원본

순차처리

병렬처리

```
Finish
Average Thread Num              : 2
Average Sequential Runtime      : 0.009000
Average Parallel Runtime        : 0.005600
Average Performance             : 60.714268
```

```
Finish
Average Thread Num              : 8
Average Sequential Runtime      : 0.008000
Average Parallel Runtime        : 0.005200
Average Performance             : 53.846176
```

```
Finish
Average Thread Num              : 4
Average Sequential Runtime      : 0.008700
Average Parallel Runtime        : 0.004700
Average Performance             : 85.106392
```

```
Finish
Average Thread Num              : 16
Average Sequential Runtime      : 0.009700
Average Parallel Runtime        : 0.007600
Average Performance             : 27.631590
```

6000x4000 이미지(PTHREAD)

원본

순차처리

병렬처리

```
Finish
Average Thread Num              : 4
Average Sequential Runtime      : 2.963000
Average Parallel Runtime        : 1.003400
Average Performance             : 195.295975
```

```
Finish
Average Thread Num              : 64
Average Sequential Runtime      : 2.977500
Average Parallel Runtime        : 0.909000
Average Performance             : 227.557755
```

```
Finish
Average Thread Num              : 16
Average Sequential Runtime      : 2.981400
Average Parallel Runtime        : 0.967300
Average Performance             : 208.218765
```

```
Finish
Average Thread Num              : 512
Average Sequential Runtime      : 2.941900
Average Parallel Runtime        : 1.062000
Average Performance             : 177.015030
```

10368x7776 이미지(PTHREAD)

원본

순차처리

병렬처리

```
Finish
Average Thread Num              : 4
Average Sequential Runtime      : 10.613199
Average Parallel Runtime        : 3.868500
Average Performance             : 174.349182
```

```
Finish
Average Thread Num              : 128
Average Sequential Runtime      : 10.580100
Average Parallel Runtime        : 3.416900
Average Performance             : 209.640335
```

```
Finish
Average Thread Num              : 64
Average Sequential Runtime      : 10.556500
Average Parallel Runtime        : 3.598900
Average Performance             : 193.325760
```

```
Finish
Average Thread Num              : 512
Average Sequential Runtime      : 10.565700
Average Parallel Runtime        : 3.681400
Average Performance             : 187.002228
```

256x256 이미지(CUDA)

원본

병렬처리

```
Finish
Block per Grid                  :  1
Thread per Block                :  1024
Average Parallel Runtime        : 0.293000
계속하려면 아무 키나 누르십시오 . . .
```

```
Finish
Block per Grid                  :  256
Thread per Block                :  256
Average Parallel Runtime        : 0.233000
계속하려면 아무 키나 누르십시오 . . .
```

```
Finish
Block per Grid                  :  5
Thread per Block                :  1024
Average Parallel Runtime        : 0.222000
계속하려면 아무 키나 누르십시오 . . .
```

결과 분석

- Sobel edge Detect

  sobel dege detect의 특성상, 데이터간의 의존성이 적기 때문에, 이를 병렬처리 할 시 순차처리에 비해 높은 성능을 보인다.

- 쓰레드의 수

  Pthread와 CUDA를 사용하여 병렬처리 할 때, 적절한 수의 쓰레드를 생성해야 한다.

- 쓰레드간의 동기화

  Pthread와 CUDA 사용 하여 병렬처리 할 때, 실행시간의 개선을 위해서 mutex(pthread), semaphore(pthread) syncthreads(CUDA) 등의 동기화의 사용을 최소화해야 한다.

Q&A