

Paralelizam i Konkurentnost

Izvješće o projektu optimiziranje programa gocr pomoću cuda platforme

Profesor: Josip Knezović

Autor izvješća: Andrija Nakić

7.2.2021.

Opis projekta

Projekt koji sam odabrao je ubrzavanje programa gocr (<https://github.com/SureChEMBL/gocr>) pomoću cuda platforme na linux operacijskom sustavu.

Sažetak rada na projektu

```
CUDAOBJJS = cuda-pgm2asc.o
```

 Slika 1.1

```
default: $(CUDAOBJJS) all
```

 Slika 1.2

```
%.o: %.cu  
    nvcc -dw -I../include $(DEFS) $< -o $@
```

 Slika 1.3

```
$(PROGRAM): lib$(PGMASCLIB).a gocr.o  
$(PROGRAM): $(LIBOBJJS)$(CUDAOBJJS) gocr.o  
# make it conform to ld --as-needed  
$(CC) -o $@ $(LDFLAGS) gocr.o ./lib$(PGMASCLIB).a $(LIBS)  
$(NVCC) -o $@ $(LDFLAGS) gocr.o $(LIBOBJJS)$(CUDAOBJJS) $(LIBS)  
# if test -r $(PROGRAM); then cp $@ ../bin; fi
```

 Slika 1.4

Projekt se prevodi tako da se prvo prevedu sve .cu datoteke u .o objekt datoteke pomoću nvcc prevoditelja (Slika 1.3). Nakon toga se prevedu sve .c datoteke pomoću gcc prevoditelja. Na kraju se sve nastale .o datoteke povezuju pomoću nvcc prevoditelja (Slika 1.4).

```
C pgm2asc.c
```

 Slika 1.5

```
compare_unknown_with_known_chars( pp, job->cfg.mode);
```

Slika 1.6

Mjerenjem vremena ustanovio sam gdje se odvija većina posla (Slika 1.5 i 1.6).

```

TIME_START

for_each_data (&(job->res.boxlist)) {
    box2 = (struct box *)list_get_current (&(job->res.boxlist));
    ii++;
    dist = 1000;

    if (box2->c == UNKNOWN || (box2->num_ac > 0 && box2->wac[0] < 97)) {
        if (box2->y1 - box2->y0 > 4 && box2->x1 - box2->x0 > 1) { // no dots!

            box4 = (struct box *)list_get_header (&(job->res.boxlist));
            dist = 1000;
            bc = UNKNOWN;

            for_each_data (&(job->res.boxlist)) {
                box3 = (struct box *)list_get_current(&(job->res.boxlist));
                wac=((box3->num_ac>0)?box3->wac[0]:100);

                if (box3 == box2 || box3->c == UNKNOWN || wac < job->cfg.certainty) continue;
                if (box2->y1 - box2->y0 < 5 || box2->x1 - box2->x0 < 3) continue;
                d = distance(pp, box2, pp, box3, cs);

                if (d < dist) {
                    dist = d;
                    bc = box3->c;
                    box4 = box3;
                }
            } end_for_each (&(job->res.boxlist));

            if (dist < 10) {
                if (box4->num_ac > 0) ad = box4->wac[0];
                else
                    ad = 97;

                ad -= dist;
                if (ad < 1) ad = 1;
                setac (box2, (wchar_t)bc, ad);
                ii++;
            } // limit as option???

            // => better max distance('e','e') ???

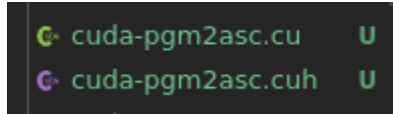
            progress (ii,pc);
        }
    }
} end_for_each (&(job->res.boxlist));

TIME_STOP

```

Slika 1.7

Na slici 1.7 je dio koda za koji je potrebno najviše vremena. Možemo uočiti dvije jednake ugnježdene for petlje i funkciju "distance". Moja ideja je bila pozvati $n * n$ instanci funkcije koja bi izvršila dio koda označen zelenom bojom. (n je broj iteracija for petlji u sekvencijalnom kodu)



Slika 1.8

Napravio sam dvije nove datoteke gdje će biti cuda kod (slika 1.8).

```
struct return_element *deviceFuncall (job_t *job, pix *pp)
```

Slika 1.9

```
cudaMalloc (&boxArr_d, n * sizeof (box_d));
cudaMalloc (&jobArr_d, sizeof (struct job_d));
cudaMalloc (&pixArr_d, sizeof (pix));
cudaMalloc (&returnArr_d, n * sizeof (struct return_element));
cudaMalloc (&treeArr_d, sizeof (tree));

cudaMemcpy (boxArr_d, boxArr, n * sizeof (box_d), cudaMemcpyHostToDevice);
cudaMemcpy (jobArr_d, jobArr, sizeof (struct job_d), cudaMemcpyHostToDevice);
cudaMemcpy (pixArr_d, pixArr, sizeof (pix), cudaMemcpyHostToDevice);
cudaMemcpy (treeArr_d, tree, sizeof (tree), cudaMemcpyHostToDevice);
cudaMemcpy (returnArr_d, returnArr, n * sizeof (struct return_element), cudaMemcpyHostToDevice);

dim3 threadsPerBlock (16, 16);
dim3 numBlocksTemp (0, 0);

if ((n / threadsPerBlock.x) * threadsPerBlock.x < n) {
    numBlocksTemp.x = n / threadsPerBlock.x + 1;
}
if ((n / threadsPerBlock.y) * threadsPerBlock.y < n) {
    numBlocksTemp.y = n / threadsPerBlock.y + 1;
}

dim3 numBlocks (numBlocksTemp.x, numBlocksTemp.y);
```

Slika 1.10

```
deviceFunc <<<numBlocks, threadsPerBlock>>> (n, boxArr_d, jobArr_d, pixArr_d, returnArr_d, treeArr_d);
cudaDeviceSynchronize ();
```

Slika 1.11

```
cudaMemcpy (returnArr, returnArr_d, n * sizeof (struct return_element), cudaMemcpyDeviceToHost);
```

Slika 1.12

Funkcija "deviceFuncCall" (Slika 1.9) će se izvršiti umjesto sekvencijalnog koda sa slike 1.7. Na slici 1.10 se vidi alociranje i kopiranje memorije na uređaj (grafičku karticu) i definicije broja blokova u gridu i broja dretvi u bloku. Na slici 1.11 se vidi pozivanje kernel funkcije "deviceFunc" i funkcije "cudaDeviceSynchronize" zbog koje host (cpu) čeka da sve dretve na uređaju završe sa izvođenjem. Na slici 1.12 se vidi kopiranje niza povratnih podataka na host. Slike 1.10 – 1.12 su isječki koda iz funkcije "deviceFuncCall".

```
__global__
void deviceFunc (int n, box_d *boxArr, struct job_d *
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
```

Slika 1.13

```
__device__
int distance_d( pix *p1, box_d *box1
{
```

Slika 1.15

```
int *dist = &(returnArr[i].dist);
int *j_max = &(returnArr[i].j_max);
```

Slika 1.16

```
d = distance_d (pp, box2, pp, box3, job, tree);
atomicMin (dist, d);
__syncthreads ();

if (d == *dist) {
    atomicMax (j_max, j);
}

__syncthreads ();

if (j == *j_max) {
    returnArr[i].box2 = *box2;
    returnArr[i].box4 = *box3;
    returnArr[i].bc = box3->c;
}
```

Slika 1.14

Isječci koda sa slika 1.16 i 1.14 su dio funkcije "deviceFunc".

Na slici 1.13 se vidi deklaracija funkcije "deviceFunc" kao `__global__` (da bi se mogla pozvati sa hosta i izvršiti na uređaju) i određivanje "i" i "j" varijabli za svaku dretvu. Svaka dretva sa istom vrijednosti "i" pohranjuje svoju lokalnu varijablu "d" na adresu pokazivača dist (Slika 1.16).

Pohranjivanje se obavlja pomoću atomične funkcije `atomicMin` koja na danu adresu pohranjuje novu vrijednost ako je manja od one na adresi. Prije određivanja "j_min" vrijednosti potrebno je pričekati da sve dretve završe sa pohranom što se postiže funkcijom `__syncthreads`. Funkcija "distance_d" (Slika 1.15) je deklarirana kao `__device__` jer se nikad ne poziva sa hosta.

Zaključak

```
nakic@nakic-dekstop:~/Documents/FER/PiK/gocr$ time src/gocr -i examples/test_image.png -o test.txt
=====
took 146606 us
=====

=====
took 501843 us
=====


=====
took 1074679 us
=====

=====
took 573926 us
=====

=====
took 2831114 us
=====

# ... found DOUBLE_LOW_9_QUOTATION_MARK
=====
took 206738 us
=====

real    0m5.586s
user    0m5.595s
sys     0m0.030s
```



Slika 1.17

```
nakic@nakic-dekstop:~/Documents/FER/PiK/gocr$ time src/gocr -i examples/test_image.png -o test.txt
=====
took 146355 us
=====

=====
took 500831 us
=====

=====
took 1091884 us
=====

=====
took 572378 us
=====

=====
gpu compute

gridDim 262, 262
blockDim 16, 16
took 2455242 us ←
=====

device to host memcpy

took 120 us
=====

# ... found DOUBLE_LOW_9_QUOTATION_MARK
=====
took 200702 us
=====

real    0m5.333s
user    0m5.227s
sys     0m0.106s
```

Slika 1.18

Slika 1.17 – izvođenje bez ubrzanja

Slika 1.18 – izvođenje sa ubrzanjem

Vrijeme izvođenja dijela koda koji je ubrzan je označeno strelicom.

specifikacije:

cpu – intel i7 4790k

gpu – nvidia gtx 970

Ubrzanje od 15% na ubrzanom dijelu koda je zanemarivo s obzirom da gpu ima cca. 1600 cuda jezgri, a za izvođenje na cpu je korištena jedna

jezgra. Zaključujem da grafičke kartice nisu optimizirane za bilo koje operacije.