Heterogén SoC rendszerek házi feladat

Batcher Odd-Even-Merge algoritumus skalár megvalósítás

Nyiri Levente

2025 Október

1. Algoritmus

Batcher Odd-Even-Merge algoritmusa először elfelezi a tömböt két részre, ezeket rendezi, külön összehasonlítja a páros elemeket a párosokkal, a páratlanokat a páratlanokkal, és végül a szomszédos elemeket egymással.

Ez egy rekurzív algoritmus. Két részből áll: egy olyan algoritmus ami a két rendezett felet kialakítja (sort), és egy olyan ami ezeket a rendezett feleket felhasználva rendez tovább (merge).

1.1. sort

Felezzük a tömbünket, meghívjuk az első és a második felére is önmagát. Amikor az átadott tömb hossza 1 lesz, akkor elérjük a bázis esetet, és a call stack-ben vissza lépkedve elkezdjük meghívni a merge függvényt.

1.2. merge

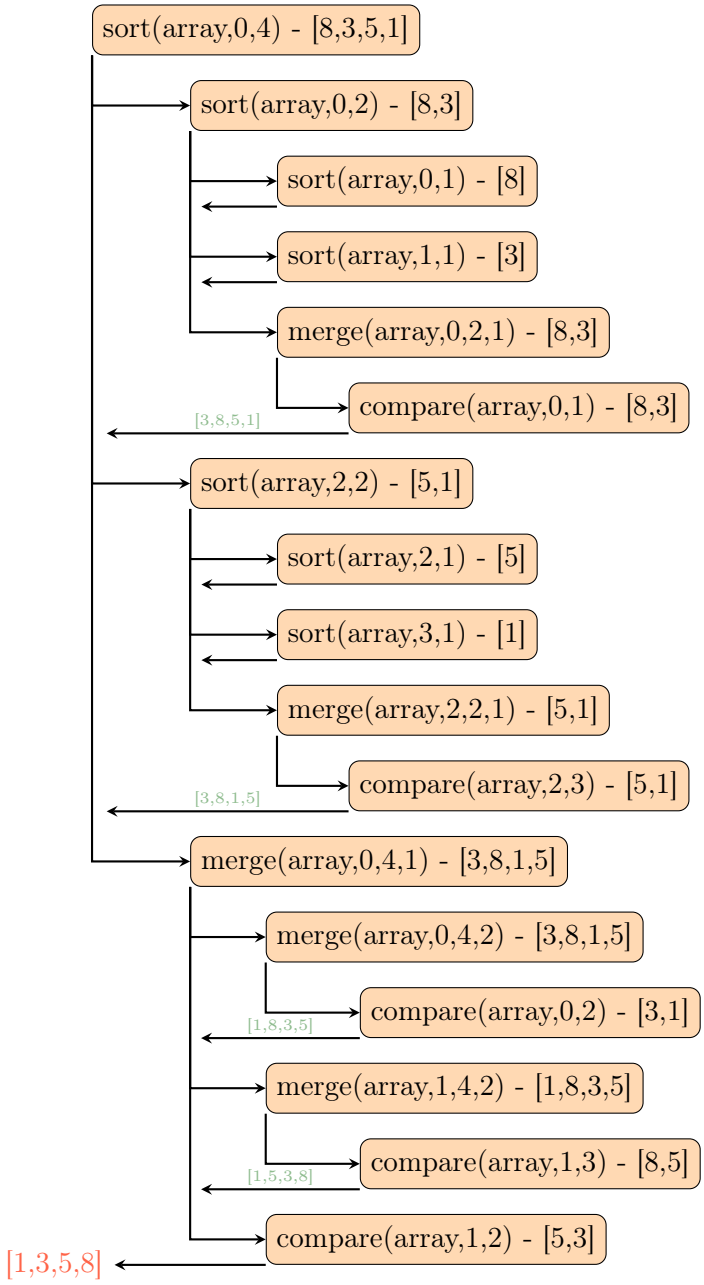
A merge függvény végez igazából minden rendezést a compare hívásával. Minden hívásnál átadunk neki egy r paramétert is (első hívásnál $r = 1$), ezzel határozzuk meg az összehasonlításoknak a lépésközét. Amikor belépünk a függvénybe először ennek a kétszeresét elmentjük egy változóba ($m = r * 2$).

Ha m kisebb mint ahány elemünk van az átadott tömbben, akkor még további rekurzív hívásokat végzünk. Először meghívjuk r helyett m-el, majd ismét, csak a kezdőpontot r-el eltolva. Ha $m =$ a tömb elemszámával, akkor összehasonlítunk kettő r távolságra lévő elemet. Miután elértük a bázis esetet, és visszelépkedünk a call stackben, egy for loopban hasonlítjuk össze a szomszédos elemeket ($16 <=$ tömböknél nem csak a végső összehasonlításban van szerepe a loopnak).

1.3. compare

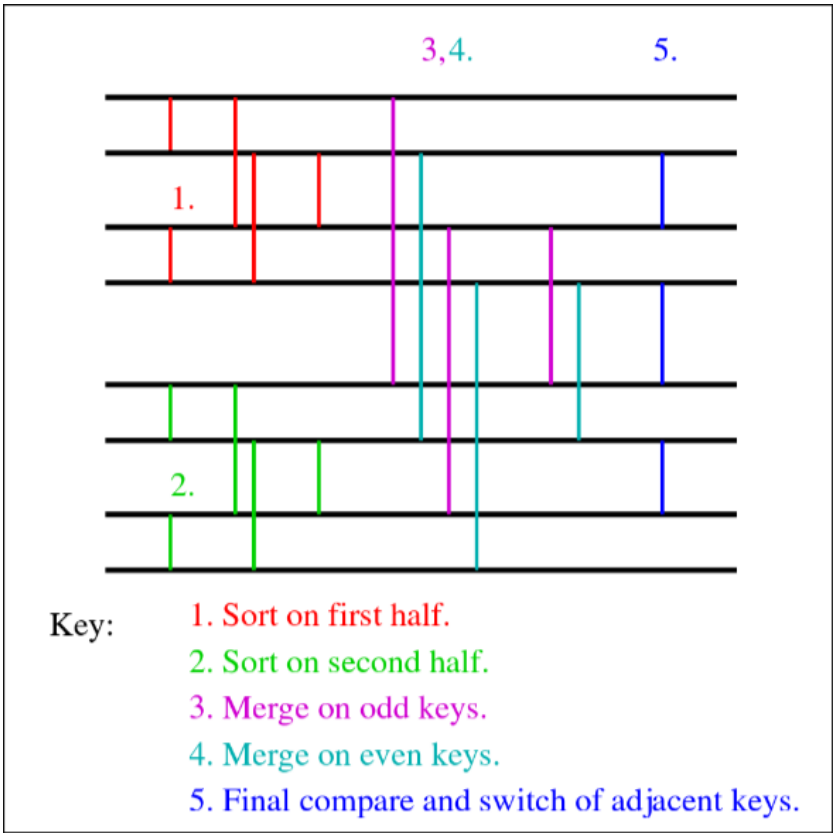
Összehasonlítja és megcseréli az átadott indexű elemeket.

A működést legjobban rekurzív algoritmusoknál szerintem call stack-el lehet szemléltetni. Egy 4 elemű tömb rendezésére egy példa az 1. ábrán látható.



1. ábra. Call stack

Egy 8 elemű tömbre egy másik vizuális reprezentáció [1] a 2. ábrán látható.



2. ábra. Batcher reprezentálás 8 inputra

2. 5x5-ös medián szűrő megvalósítás

A megvalósításhoz minden sorban komponensenként lépkedve elmentek egy 5x5-ös ablakot, erre futtatom az algoritmust, és az így rendezett tömbnek a 12-ik (medián) elemét adom vissza részeredményként. Mielőtt átadom a rendező függvénynek a tömböt kibővítem 255-ös értékekkel (8 bites komponensek) 25-ről 32 elemre, mivel csak 2 hatvány nagyságú tömbökre működik az algoritmus.

```
#include "batcher.h"
#include <stdio.h>
#include "stdbool.h"
#include <cstring>

void compare(uint8_t* array, int i, int j)
{
  if (array[i]>array[j])
  {
    uint8_t tmp = array[i];
    array[i] = array[j];
    array[j] = tmp;
  }

}

void merge(uint8_t* array, int start, int length, int r)
{
  int m=r*2;
  if(m<length)
  {
    merge(array, start, length, m);
    merge(array, start+r, length, m);
    for (int i=start+r; i+r<start+length; i+=m)
      compare(array, i, i+r);
  }
  else
    compare(array, start, start+r);
}

void sort(uint8_t* array, int start, int length)
{

  if(length>1)
  {
    int m = length/2;
    sort(array, start, m);
    sort(array, start+m, m);
    merge(array, start, length, 1);
  }

}

void media_filter_scalar(int imgHeight, int imgWidth, int imgWidthF,
        uint8_t *imgSrcExt, uint8_t *imgDst)
{
  for(int row=0; row<imgHeight; row++)
  {
    for(int col=0; col<imgWidthF; col++)
```

```
50      {
51        uint8_t r_vals[25], g_vals[25], b_vals[25];
52        int rd_offset;
53
54        for(int fy=0; fy<5; fy++)
55        {
56          for(int fx=0; fx<5; fx++)
57          {
58            rd_offset = ((row+fy)*imgWidthF + (col+fx))*3;
59            int idx = fy*5+fx;
60
61            r_vals[idx] = *(imgSrcExt + rd_offset + 0);
62            g_vals[idx] = *(imgSrcExt + rd_offset + 1);
63            b_vals[idx] = *(imgSrcExt + rd_offset + 2);
64          }
65        }
66        uint8_t padded_r[32], padded_g[32], padded_b[32];
67        memcpy(padded_r, r_vals, 25);
68        memset(padded_r + 25, 255, 7);
69        memcpy(padded_g, g_vals, 25);
70        memset(padded_g + 25, 255, 7);
71        memcpy(padded_b, b_vals, 25);
72        memset(padded_b + 25, 255, 7);
73
74        sort(padded_r,0,32);
75        sort(padded_g,0,32);
76        sort(padded_b,0,32);
77
78        int wr_offset;
79        wr_offset = row*imgWidth*3 + col*3;
80
81        *(imgDst + wr_offset + 0) = padded_r[12];
82        *(imgDst + wr_offset + 1) = padded_g[12];
83        *(imgDst + wr_offset + 2) = padded_b[12];
84
85      }
86    }
87  }
```

1. Listing. 5x5 medián szűrő C-kód

A működést teszteltem is egy egyszerű python scripttel, ugyanazt az eredményt adja mint a referencia.

```python
from PIL import Image, ImageChops
import sys, getopt
import numpy as np

if len(sys.argv) != 3:
    print("Usage: python test.py image1.png image2.png")
    sys.exit(1)

img1 = Image.open(sys.argv[1])
img2 = Image.open(sys.argv[2])

diff = ImageChops.difference(img1, img2)

diff_array = np.array(diff)

different_pixels = np.count_nonzero(diff_array.any(axis=2))

print(f"Differing pixels: {different_pixels}")
total_pixels = diff_array.shape[0] * diff_array.shape[1]
print(f"Total pixels: {total_pixels}")
print(f"Percentage different: {different_pixels / total_pixels *
    100:.2f}%")
```

2. Listing. Python összehasonlító

Hivatkozások

[1] „Batcher Odd-Even-Mergesort”. cím: `https://math.mit.edu/~shor/18.310/batcher.pdf`.