Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Osztott rendszerek specifikációja és implementációja

IP-08bORSIG

Dokumentáció a 3. beadandóhoz

Abonyi-Tóth Ádám
DKC31P

2017. december 29.

1. Kitűzött feladat

Egy előre megadott fájlban képek szöveges reprezentációját találhatjuk (pixelenként RGB módon).

A program parancssori paraméterként kapja meg az alábbiakat:

- A képek átméretezési arányát %-ban. (pl. 50 - ekkor 50%-ra, azaz felére kell csökkenteni az összes képet. 25 esetén negyed-méretet kapnánk, etc.)
- Annak a fájlnak a neve, a képek primitív leírását tartalmazza. (pl. `'pictures.txt'`)
- A kimeneti fájl neve (pl. `'picross.quiz'`)

Egy képen végrehajtandó transzformáció a következő lépések kompozíciójával áll elő:

- A fájlból (első param.) beolvasott képeket először a megadott arányban át kell méretezni.
- Ezek után az így kapott kisebb képek színeit kell leképezni az előre megadott 8 szín valamelyikére.
- Ezt követően az így kapott ábrákban minden sorra és oszlopra ki kell számolni, hogy egymás után hány azonos színű pixelt láthatunk (de nem szimplán azt, hogy az adott sorban/oszlopban hány különböző szín található).

A kapott eredményeket (a méretezett és megfelelő színre konvertált képeket és a hozzájuk tartozó címkéket) írjuk ki a kimeneti fájlba (3-ik paraméter)!

2. Felhasználói dokumentáció

2.1. Rendszer-követelmények, telepítés

A program futtatásához szükséges a PVM rendszer megléte. Telepítéséhez a négy bináris állomány, a *master*, *first*, *second* és a *third* megfelelő helyen való elhelyezése szükséges.

2.2. A program használata

A program futtatásához előbb lépjünk be PVM-be, és inicializáljuk tetszés szerint a bladeeket. Az indítás a PVM-en belül a következő paranccsal történik:
`spawn -> master rate input.txt output.txt`,
ahol `rate` az átméretezés mértéke százalékban, `input.txt` a bemeneti fájl neve, `output.txt` pedig a kimeneti fájl neve a felhasználó könyvtárához képest.
Figyeljünk az inputfájlban található adatok helyességére és megfelelő tagolására, mivel

az alkalmazás külön ellenőrzést nem végez erre vonatkozóan. Futás után a paraméterben megadott fájl tartalmazza a kapott eredményt.

3. Fejlesztői dokumentáció

3.1. Megoldási mód

A programot az *adatcsatorna tételére* vezetjük vissza. A kódot logikailag három egységre osztjuk:

- *Főfolyamat:* A szülőfolyamat feladata a bemenet és a kimenet kezelése és az adatcsatorna felállítása.
- *Gyermekfolyamatok:* A 3 gyermek feladata az egyes függvénykomponensek (f_1, f_2 és f_3) végrehajtása. ~~És hogy megvédjék a Földet az Angyalok támadásaitól és a Third Impact-től.~~
- *Model:* A `pvm_utils.hpp`, `image.hpp` és `resize.hpp` fájlok szerepe a folyamatok közös típusainak és segédfüggvények különválasztása a tesztelhetőség és a karbantarthatóság érdekében.

3.2. Implementáció

A feladat alapvető típusai az `Image` (RGB kép reprezentációja), a `Image3bit` (3 bites kép ábrázolása), és a `ColorTag = std::vector<std::vector<int>>` (a címkék reprezentációja).

A `master.cpp` fájlban van a főprogram megvalósítása. A `main` függvény beolvassa a bemeneti fájlból a képeket, majd a PVM segítségével létrehoz 3 gyermeket, és elküldi nekik az adatcsatorna létrehozásához szükséges információkat. Ezután amíg van adat, fogadja a megoldásokat a harmadik gyermektől.

A gyermekek egy általános sémát követnek, amit a `pvm_utils::ChannelChild` típusban rögzítünk. Ez az osztály biztosítja azt a működést, hogy ha adott egy függvényt, akkor amíg a csatornán van adat, addig leveszi a bemenetet a csatornáról, alkalmazza rá a függvényt, és továbbítja az eredményt a következő csatornán. Ezt az osztályt használva az egyes gyermekeknél elég megadni magát az f_i függvényt.

Az `first` feladata az átméretezés, bemenete az arány és a forráskép (`std::tuple<int,Image>`), kimenete pedig a lekicsinyített kép (`Image`). A megvalósítás *Divide & Conquer* stratégiát használ, ahol a kép részeit (`ImgPart`) vagy további négy részre osztjuk (`slicing::slice`), elvégezzük új szálon a számításokat, és egyesítjük őket

(`slicing::combine`), vagy ha elég kicsik, akkor kiszámoljuk az eredeti kép alapján a megfelelő pixel értékeket.

A `second` feladata a színek (`Color`) konvertálása 3 bites tartományba (`Color3bit`), így a bemeneti `Image`-et *Task farm* stratégiával soronként átalakítva visszaadja az `Image3bit` eredményt.

A `third` gyermek a hárombites képre előállítja a címkéket (`ColorTag`). Ez soronként és oszloponként szintén *Task farm* segítségével történik. Amíg a task farm részfeladatai számolnak, a főfolyamat visszaalakítja a hárombites képet `Image` típusúra.

3.3. Fordítás menete

A fordítás a tantárgy honlapjáról letöltött `Makefile.aimk` fájl és az `aimk` eszköz segítségével történik. A makefile tartalmából kiderül, hogy a kódban használhatjuk a C++11-es szabványos elemeket. A modulokat úgy jelöljük ki fordításra, hogy hozzáfűzzük a makefile első sorához a nevüket, kiterjesztés nélkül.

3.4. Tesztelés

A program helyességének tesztelése szekvenciális módon történt, lokálisan. Ezt főként az tette lehetővé, hogy minden PVM-specifikus kód a `pvm_utils` névtérben van elkülönítve, és független a többi egység működésétől, így PVM-et nélkülöző rendszerben is tesztelhető volt az alkalmazás.

Ezt követte a kommunikáció tesztelése az Atlaszon, ahol a program szintén helyesen működött a példabemenetekre.

Beláthatjuk, hogy a pvm rendszerben háromnál több blade használata már nem fogja tovább gyorsítani a futtatást, mivel pvm-es gyermek csak 3 van, 3 aktív blade-nél viszont kell ahhoz, hogy mindegyik gyermek a lehető legtöbb erőforráshoz jusson.