Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Osztott rendszerek specifikációja és implementációja

IP-08bORSIG

Dokumentáció a 3. beadandóhoz

Nagy Richárd Tibor
GWSAZV

2017. december 19.

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')

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 a PVM keretrendszerre van szükség, annak megléte nélkül nem használható. Külön telepítésre nincsen szükség.

2.2. A program használata

A programot taszkként indíthatjuk el PVM-ből. Három paramétert vár: a tömörítés mértékét, a bemeneti adatokat tartalmazó fájlt, valamint a kimeneti fájl nevét. Egy lehetséges futtatása az alábbi módon valósulhat meg:

pvm

spawn -> master 50 input.txt output.txt

A fent említett szöveges fájlokat a home könyvtárban kell elhelyezni. A bemeneti fájl formátumának helyessége nincs ellenőrizve, hibás formázás esetén az alkalmazás megfelelő működése nem garantált. A futás végeztével a paraméterként megadott kimeneti fájlban található az eredmény.

3. Fejlesztői dokumentáció

3.1. Megoldási mód

A feladatot adatcsatornával oldjuk meg, PVM környezetben. A párhuzamos programunk négy különálló C++ programból áll (master, first, second, third), melyek az egyes részfeladatokat végzik. Futtatás során ezek kapcsolatba lépnek: PVM-en keresztül a master indítja a többieket és adja át a kezdetben szükséges információkat, ezen felül a first rekurzívan önmagából is indíthat példányokat. A részprogramok sorban üzeneteket is küldhetnek egymásnak. A master végzi ezen felül a beolvasást és a kiírást, az utóbbihoz az adatokat a third-től kapja.

3.2. Implementáció

A forráskódok mérete és komplexitása nem indokolja azok a már meglévőknél több fájlba való szétbontását, így azok teljes egészükben a `master.cpp`, a `first.cpp`, a `second.cpp` és a `third.cpp` nevű állományokban találhatóak. Az adatokat `std::vector`-okban tároljuk, a második és harmadik lépésben a TaskFarm-ok `std::future`-rel vannak megvalósítva. Az új taszkok indítását PVM-ben a `pvm_spawn()`, új szálakét C++11-ben pedig az `std::asycn()` függvény végzi. A taszkok közötti kommunikációt a `pvm_send()` és a `pvm_recv()` alprogramok, az üzenetek be- és kicsomagolását pedig a `pvm_pk*()` és `pvm_upk*()` mintájú függvények hajtják végre.

3.3. Visszavezetés az adatcsatorna tételére

Az absztrakt tétel állapotterében szereplő X_0-nak a bemeneti fájl, az X_{n+1}-nek pedig a kimeneti fájl felel meg. A D adatsorozat elemei az inputfájlban szereplő képek, az output fájl végső állapota $F(D)$-vel egyezik meg. Az F függvény $n = 3$ db függvény kompozíciójaként áll elő ($F = f_1 \circ f_2 \circ f_3$), ahol f_1, f_2 és f_3 rendre a first, second és third nevű programok. Ezek végzik a feladatban leírt átalakításokat.

3.4. Fordítás menete

A fordításhoz elengedhetetlen egy `C++11` szabványt támogató fordítóprogram a rendszeren. Ehhez használhatjuk az *MSVC*, *g++* és *clang* bármelyikét. A fordításhoz célszerű létrehozni egy makefile állományt, ami automatikusan elvégzi a fordítást, ezek eredményének áthelyezését a megfelelő mappákba, a linkelést, valamint a takarítást. Fontos, hogy `-std=c++11` kapcsolóval fordítsunk, mert lehetséges hogy alapértelmezés szerint a fordítóprogram még a régi, C++98-as szabványt követi, melyben a felhasznált nyelvi elemek még nem voltak jelen.

3.5. Tesztelés

A tesztelést az ELTE Atlasz nevű számítógép klaszterén végeztem. Tesztjeim során 11 blade állt rendelkezésemre. A program tesztelését több különböző mintafájlon is elvégeztem, azok alapján a működését helyesnek ítéltem.

Elsőként az tömörítés során alkalmazott rekurzió optimális mélységét határoztam meg. Az átlagos futási idők a kapott in3.txt fájlra, 11 blade-del, 1/4-es tömörítéssel a következők:

8x8-asra darabolás: 0.925172 sec

16x16-osra darabolás: 0.269294 sec

32x32-esre darabolás: 0.0214938 sec

Más tömörítési mértékekkel az eredmények aránya ugyanilyen. A mérések alapján 32x32-nél kisebbre darabolni biztosan nem éri meg.

Ezután mértem a teljesítményt a blade-ek számától függően.

11 blade: 0.0636167 sec

8 blade: 0.0622882 sec

4 blade: 0.0554741 sec

3 blade: 0.0531694 sec

2 blade: 0.0794802 sec

1 blade: 0.1245849 sec

A teljesítmény akkor a legjobb, ha a csatorna tagjainak száma és a dolgozó blade-ek száma megegyezik.