Software Concurrent

OpenMP: Cercador d'imatges

Vicent Roig, Igor Dzinka

22/05/2015

Índex

1. Objectius

L'objectiu d'aquesta pràctica és implementar una aplicació que permeti cercar imatges fent servir l'histograma per descriure-les. En particular, l'aplicació ha d'implementar les següents funcionalitats.

1. Extracció de l'histograma de les imatges.
2. Cerca d'imatges similars a partir d'una imatge exemple.
3. Persistència en les dades extretes de la imatge.

2. Introducció

Per a la implementació de la pràctica s'ha seguit l'estructura suggerida a l'enunciat, on bàsicament l'arbre de fitxers emprat per la persistència seguirà la següent estructura:

```
['db']
  |____ ['images']
  |
  |____ ['histrograms']
  |
  |____ '.id' (hidden)
```

Inicialment es comprovarà si existeix l'arbre de directoris, en cas afirmatiu es procedirà a la generació i posterior càrrega dels corresponents histogrames a memòria RAM i a presentar la llista d'imatges per pantalla, de manera que un usuari pugui seleccionar-ne una d'elles per poder-la visualitzar. Observem que el procés de càrrega d'histogrames i llistat d'imatges ha set una de les seccions paral·lelitzades amb OMP.

Per a la realització de les proves hem fet ús del conjunt de 30607 imatges disponible a:

- http://www.vision.caltech.edu/Image_Datasets/Caltech256/

Cada cop que decidim importar noves imatges, es prendrà com a referència d'indexació la variable 'identifier', notem aquesta variable s'inicialitza cada cop que s'obre l'aplicació. Si existeix l'arbre recupera l'enter escrit al fitxer ocult '.id', altrament s'inicialitza a 1.

El fitxer de text pla utilitzat per importar noves imatges a la nostre base de dades pot ser generat de la següent forma:

```
:pwd$ cd /absolute_path_to_db/256_ObjectCategories/
:256_ObjectCategories$ find -name *.jpg >> ./path_to_project_folder/filename.txt
```

```
# OSX requires '.' as it does not infer current dir
MacBook-Pro:256_ObjectCategories user$ find . -name *.jpg >> ../filename.txt
(...) see 'example_list.txt' in order to make an idea.
```

Aquest projecte es trobarà disponible públicament al nostre repositori arribada la data d'entrega. El repositori és hospedat a BitBucket i fa ús del controlador de versions GIT. Al README.md s'exposen els detalls a tenir en compte a l'hora de executar-ho:

```
git clone https://bitbucket.org/simorgh12/sc2015.git
:pwd$ cd /sc2015
:sc2015$ git checkout practica2
```

3. Compilació del codi

El projecte ha set realitzat amb Qt Creator, el fitxer de configuració associat per poder-ho compilar esta inclòs al propi projecte amb la extensió .pro.

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-05-06T17:46:48
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Practica_OpenMP
TEMPLATE = app
INCLUDEPATH += "/usr/local/include/opencv2"

SOURCES +=  main.cpp\
            mainwindow.cpp \
            histogramManager.cpp

HEADERS  += mainwindow.h \
            histogramManager.h

LIBS += `pkg-config opencv --libs`

#QMAKE_CXXFLAGS+= -openmp
#QMAKE_LFLAGS +=  -openmp


QMAKE_CXXFLAGS += -fopenmp
QMAKE_LFLAGS   += -fopenmp

FORMS    += mainwindow.ui
```

4. Paral·lelisme

Construcció de la base de dades

Com ja s'ha comentat a la secció introductòria, al nostre programa, la base de dades es constreix a partir d'un fitxer de text, que a cada línia conté la ruta a la imatge que es vol afegir. Al utilitzar aquest model, sabem quants fitxers anirem incloure, que coincideix amb el nombre de línies que té el fitxer de càrrega.

Coneixent el número d'operacions, hem pogut realitzar la paral·lelització del procés amb una construcció en bucle **#pragma omp parallel for.** Observem que en aquest cas fer ús la construcció en bucle és més recomanable que no pas la construcció en la tasca, ja que l'*overhead* associat es menor.

Al realitzar les proves, ens hem adonat que al finalitzar la operació de construcció de la base de dades, un gran nombre de iteracions les realitzava nomes un fil. El problema es déu al *scheduling* que es feia per repartir la feina entre els fils.

Per tal de millorar el repartiment de les tasques entre el fils, a mes a mes hem afegit *un schedule dynamic*

Persistència

Hem paral·lelitzat el procés de la carrega d'una base de dades existent a la memòria. Recordem que a la carpeta de la base de dades disposem d'un fitxer ocult que conté el nombre d'imatges que es té. Això, igual que en el cas de la importació de la base de dades ha propiciat que féssim la paral·lelització en bucle.

De la mateixa forma que abans, hem vist que la finalització de la càrrega no es reparteix bé entre els threads i hem afegint un scheduling dynamic a la construcció en bucle.

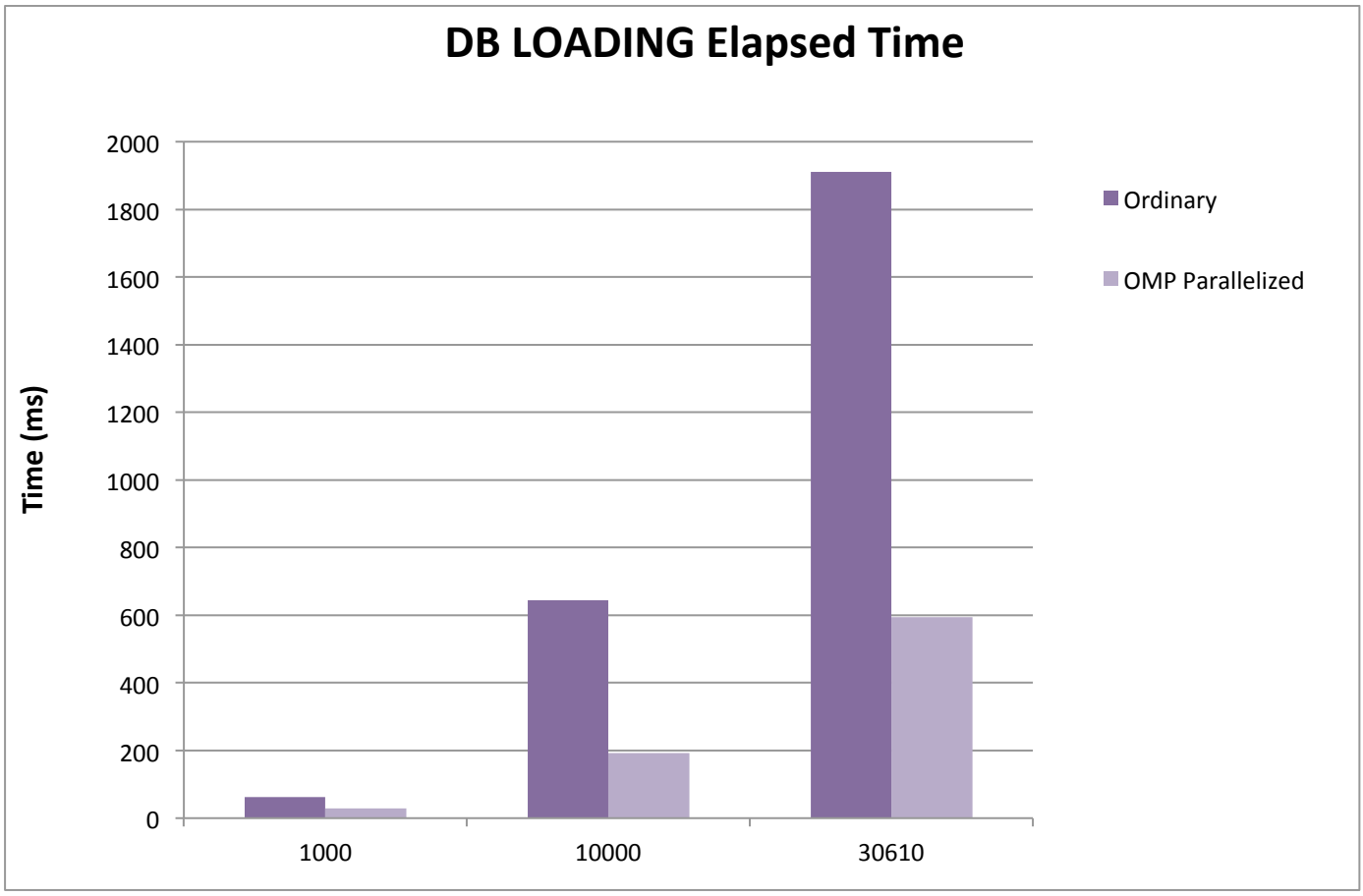
Cerca d'una imatge en una base de dades

Per a la paral·lelització de la cerca d'una imatge a la base de dades, fem la construcció en bucle per la mateixa raó que abans, coneixem el nombre de comparacions que realitzarem (comparem amb tots els arxius de la base de dades i després en seleccionem 4 millors resultats i els mostrem per pantalla).

5. Temps d'execució

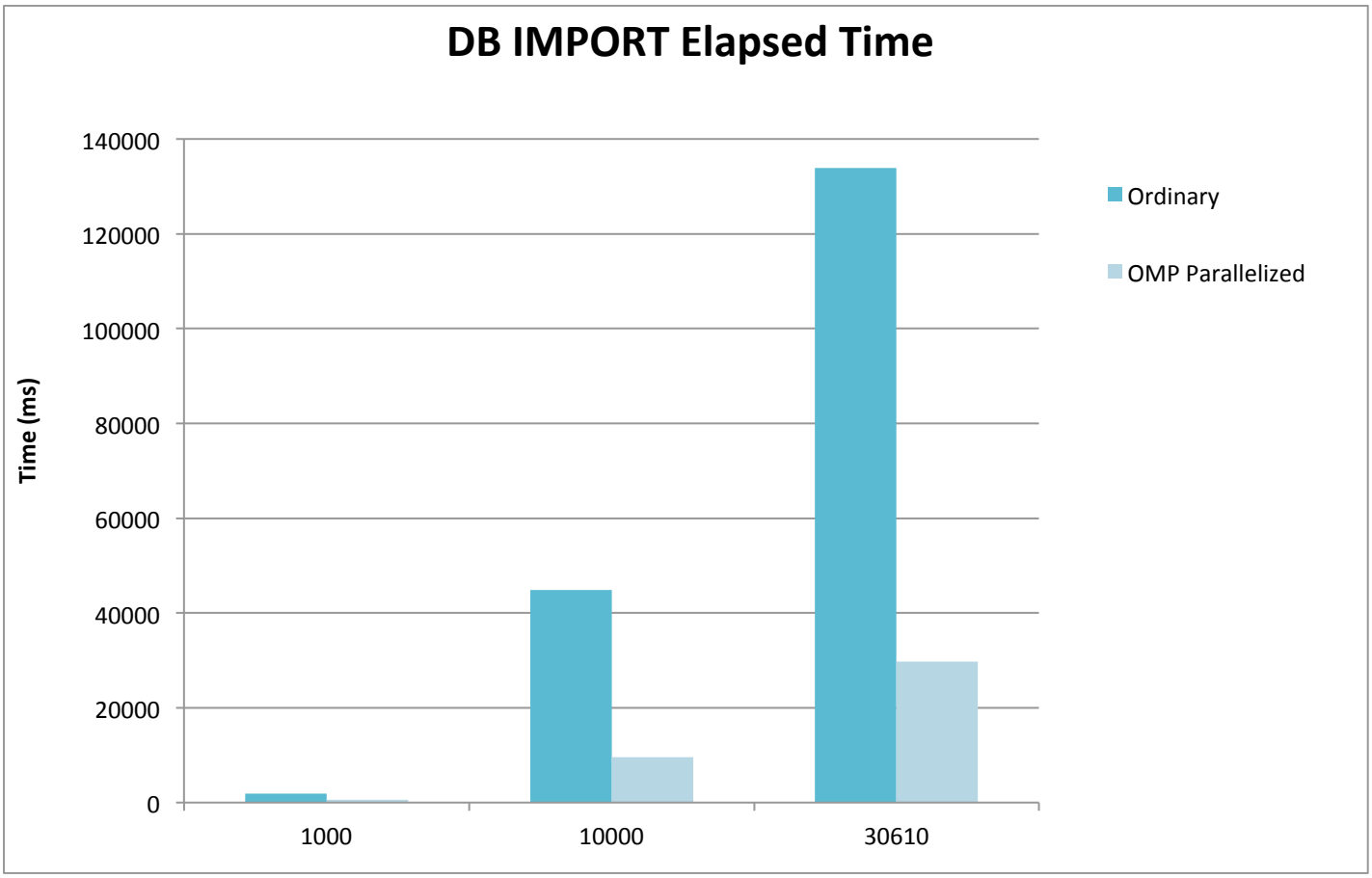
A continuació mostrem els resultats de les proves d'execució de diferents processos que hem paral·lelitzat. Hem realitzat les proves amb diferents quantitats de fitxers (**1 000**, **10 000** i **30 610** que es la mida completa de la base de dades oferta per fer proves).

La recuperació d'una base de dades existent i la seva carrega a la memòria RAM:



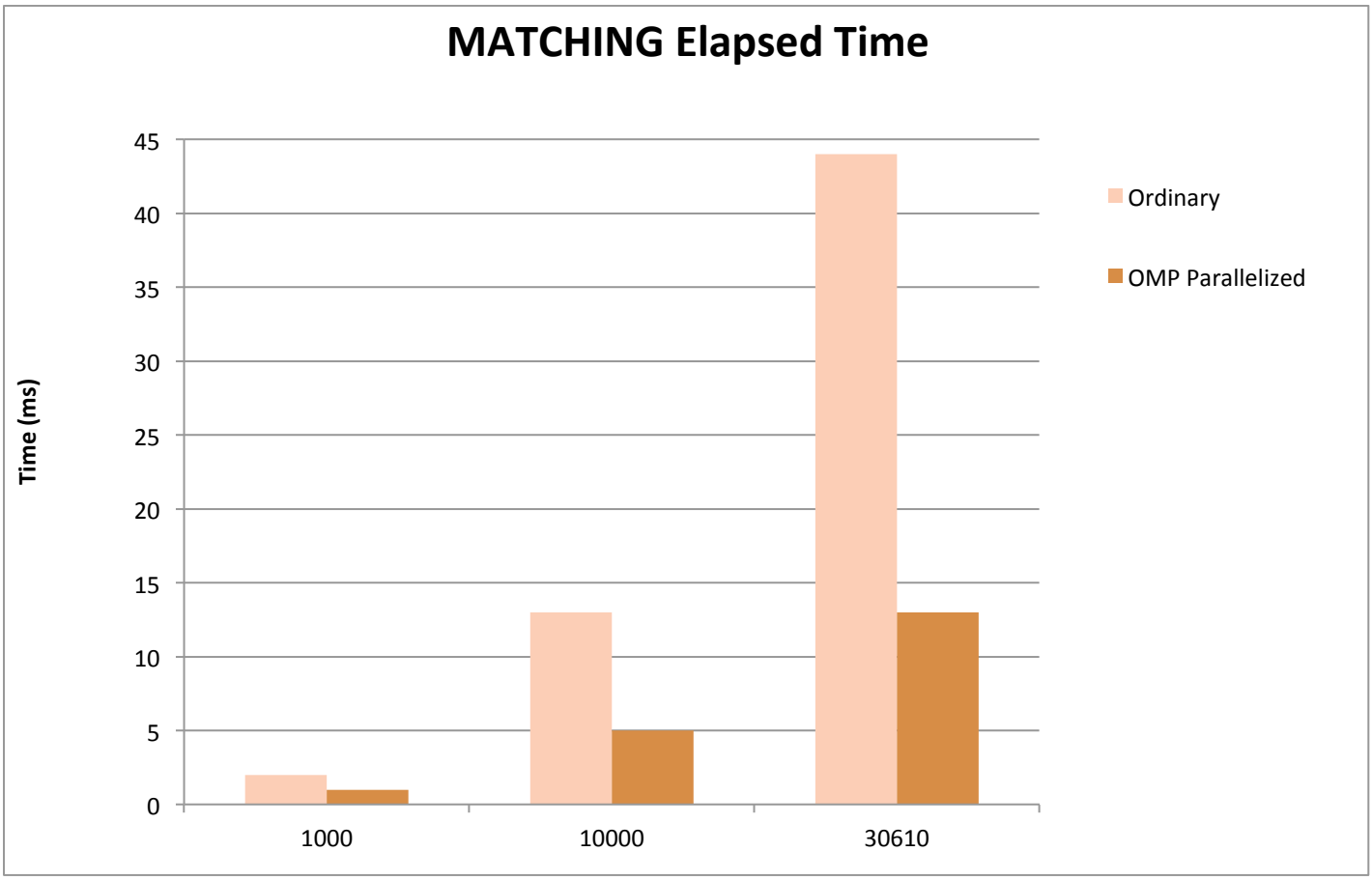
DB size	Ordinary	OMP Parallelized
1000	63	29
10000	644	193
30610	1911	595

La importació de la base de dades fent servir un fitxer de text.



DB size	Ordinary	OMP Parallelized
1000	1944	584
10000	44871	9581
30610	133870	29689

Cerca de la imatge a la base de dades mitjançant comparació d'histogrames.



DB size	Ordinary	OMP Parallelized
1000	2	1
10000	13	5
30610	44	13