IMAGine

Francesco Paolo Castiglione, Davide Iraci, Andrea Montemaggiore

Settembre 2020

Indice

1 Introduzione

IMAGine (IMAge-enGINE) è un linguaggio ideato per l'elaborazione delle immagini a trecentosessanta gradi. Il linguaggio consente le elaborazioni più comuni attraverso un processo immediato ed intuitivo. E' stato pensato e sviluppato tenendo conto di alcuni fattori chiavi quali semplicità d'uso e rapidità di apprendimento, basandosi anche su altri linguaggi già esistenti quali MATLAB, Python o software per la modifica di immagini.
Il resto della relazione è strutturato come segue: la sezione 2 illustra il contesto in cui si colloca il linguaggio proposto, nella sezione 3 vengono analizzati i dettagli implementativi, nella sezione 4 si illustrano le caratteristiche di IMAGine e degli esempi di applicazioni, infine nella sezione 5 vengono riepilogati i punti principali del lavoro svolto.

2 Stato dell'arte

Attualmente l'Image Processing avviene principalmente per mezzo di software già pronti quali Photoshop, Gimp, etc. Questi software sono stati pensati per un utilizzo da parte di amanti delle immagini e per professionisti in campo fotografico. Alcuni di questi, come il già citato Adobe Photoshop vengono proposti in una suite software chiamata "Adobe Creative 2020", la quale, nella sua ultima edizione, comprende altri applicativi per l'Image Processing di livello sempre più alto. Per lo sviluppo del progetto ci siamo principalmente basati sullo stato dell'arte dei linguaggi che permettono operazioni su immagini e sulle operazioni che permettono: da semplici rotazioni, zooming e shrinking alle più complicate operazioni di edge detection, cambio dello spazio di colori, etc;
Come linguaggio da adoperare per produrre il progetto sono stati individuati linguaggi dalla sintassi concisa ed efficace quali C, C++, Java e Python. Ognuno di detti linguaggi ha pro e contro ma tutti richiedono l'utilizzo di librerie esterne per l'implementazione delle modifiche e operazioni su immagini. C e Python sono attualmente i più veloci. C presenta una sintassi più complessa per l'utilizzo delle relative librerie mentre Python possiede numerose librerie con una sintassi di facile utilizzo ma alcune di esse, o meglio quasi tutte, non ottengono un buon risultato performando alcune operazioni più complicate come edge detection. Va sottolineato come il package "skimage" per Python sia uno dei migliori in termini di prestazioni e risultati. Per C e C++ la libreria migliore in termini di prestazioni è "LibVips" adoperata anche in questo progetto, nonostante possieda una sintassi di difficile utilizzo per via dei lunghi nomi dei metodi. L'ultima analisi si basa su Matlab, ampiamente usato, specialmente in ambito accademico. Si tratta del linguaggio per immagini per eccellenza. Alcune operazioni, come media adattiva, mediano adattivo, etc, richiedono che il programmatore abbia una conoscenza approfondita del linguaggio ed ambiente di sviluppo MATLAB, mentre per operazioni più semplici, quale applicazione di filtri di media e mediano, è sufficiente utilizzare funzioni built-in. Dallo stato dell'arte vengono dunque individuati i passaggi chiave per la creazione del linguaggio: sintassi semplici e prestazioni d'alto livello.

scipy 1.2, numpy 1.15.4, pillow	4.33	361	29

Figura 1: Risultato benchmark SciPy

3 Descrizione del progetto

In questa sezione vengono illustrate le scelte progettuali e le motivazioni che hanno portato alla scelta della libreria LibVips.

3.1 Scelte progettuali

Dal punto di vista sintattico il team si è ispirato a linguaggi dalla sintassi intuitiva quali *Javascript* , *Python* e *C#*. Le variabili vengono dichiarate senza specificare un tipo esplicito. Il valore viene semplicemente assegnato alla variabile in fase di inizializzazione della stessa o successivamente. Le immagini vengono dichiarate fornendo il path, rendendo cosi' più manegevole l'elaborazione delle immagini. Il tipo ricorsivo lista può contenere qualunque elemento, compreso un tipo lista.
Particolare attenzione è stata prestata alla *leggibilità* del codice IMAGine ed alla *semplicità d'uso* nel dominio scelto, ovvero l'elaborazione delle immagini. Il linguaggio si presta all'utilizzo da parte di utenti occasionali e presenta una veloce curva di apprendimento.

3.2 LibVips

l team in fase di analisi ricercato al fine di individuare una libreria che fosse veloce e che allo stesso tempo non usasse una alta quantità di memoria per le operazioni effettuate sulle immagini. A tal proposito il team ha valutato i risultati dei benchmarks sulle librerie. Da tale analisi è apparso chiaro che LibVips si classivica per 0.04 secondi come seconda libreria più veloce utilizzando in media ben 100 mb di memoria in meno rispetto alla prima classificata per la stessa operazione. In figura 2 viene riportata una tabella contenente delle informazioni sui benchmarks risultato dell'operazione di copia di un' immagine di 5000x5000 pixel 8-bit RGB in formato TIFF non compressa.
Si può facilmente notare che LibVips lanciato in modalità one thread o in modalità command-line rimane comunque una delle librerie più affidabili. Inoltre nella lunga lista di alternative non sono presenti esclusivamente librerie di image processing per C/C++ ma anche librerie per Python come la famosa SciPy basata su NumPy. In figura 1 vengono riportati i relativi dati.
Analizzando meglio le varie implementazioni ci si è resi conto che VipLips in modalità command-line genera un grosso traffico di dati sul disco. In figura 3 è possibile notare come LibVips (ultimo nella legenda) sia quello che utilizza meno memoria e meno tempo e quindi la miglior libreria utilizzabile.
Un'altra serie di benchmarks, illustrati in figura 4 è stata effettuata prendendo un'immagine di 10000x10000 pixel scattata da una macchina digitale sulla quale sono successivamente state applicate le seguenti operazioni : sottoposta a cambio di spazio di colori, ridimensionata, tagliata (cropping) e sottoposta ad operazione di sharpening. Tale operazione è stata effettuata su diversi processi, alcuni di livello server ad altri di uso domestico.
Si può notare che perfino su dispositivi completamente diversi come Raspberry Pi 2 il tempo per l'operazione non sia altissimo, considerando che l'utilizzo di questa libreria rimane una delle poche opzioni a disposizione dell'utente in quanto sicuramente le suite di software con interfaccia grafica, ad esempio Adobe Creative, non possono essere utilizzate agevolmente su alcuni degli ultimi processori/dispositivi in figura 4 .

Software	Run time (secs real)	Memory (peak RSS MB)	Times slower
tiffcp -s	0.11	148	
libvips C 8.8	0.15	40	1.0
lua-vips 1.1-9	0.15	49	1.0
pyvips 2.1.6	0.18	49	1.2
ruby-vips 2.0.14	0.21	49	1.4
libvips C 8.8, JPEG images	0.26	48	1.7
libvips command-line 8.8	0.33	38	2.2
Pillow-SIMD 5.3 *see 1*	0.36	230	2.4
ymagine git master 15/12/15 *see 2*	1.06	2.9	3.2 *compared to vips-c JPEG*
libvips C 8.8, one thread	0.52	33	3.5
libvips nip2 8.8	0.57	77	3.8
NetPBM 10.0-15.3	0.60	75	4
sips 10.4.4 *see 6*	0.7 *est.*	268	4.1
GraphicsMagick 1.3.28	0.64	493	4.3
ImageMagick 6.9.7-4	0.82	463	5.5
RMagick 2.16.0 *ImageMagick 6.9.7-4*	0.83	720	5.5
libgd 2.2.5 *see 2*	1.46	193	5.6 *compared to vips-c JPEG*
OpenCV 3.2	0.93	222	6.2
Imlib2 1.4.10 *see 9*	1.14	242	7.6
ExactImage 1.0.1 *see 3*	1.15	122	7.7
FreeImage 3.17 *see 7*	1.28	185	8.5

Figura 2: Risultato benchmark

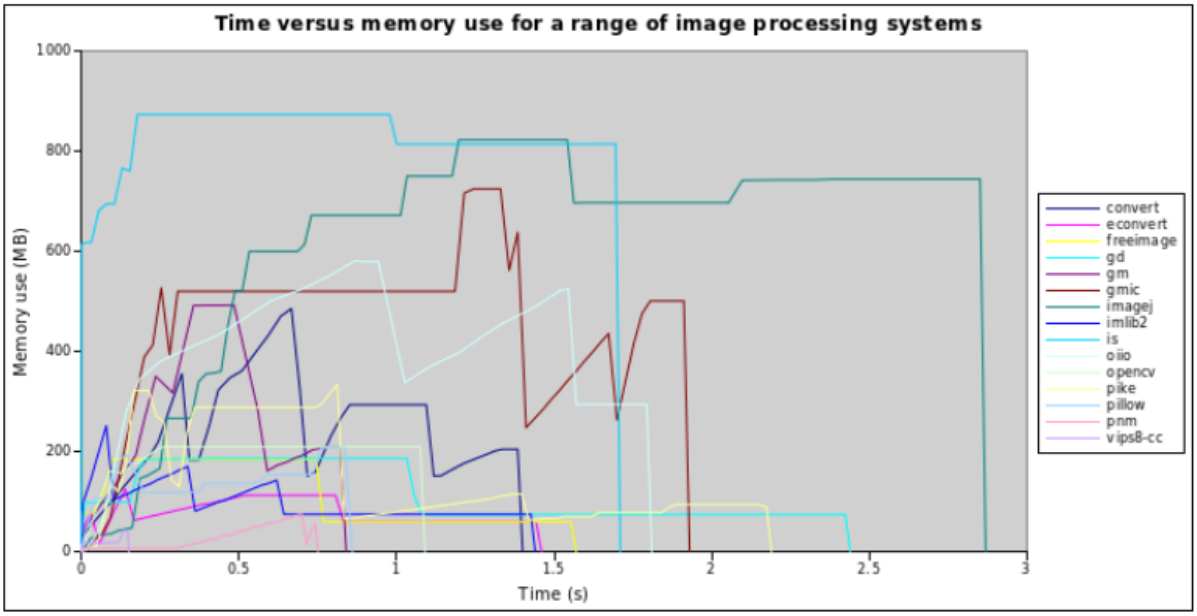

Figura 3: Risultato benchmark uso memoria

3.3 Guida all'uso

Per poter utilizzare il linguaggio in esame è necessario installare LibVips, a tal proposito è consigliato scaricare il .tar dal seguente link : `https://github.com/libvips/libvips/releases`
Prima di installare la libreria è necessario installare le seguenti dependencies:
`build-essential, pkg-config, glib2.0-dev, libexpat1-dev, expat`
Per ognuna di esse scrivere su terminale Unix "`sudo apt-get install XXX`". Dopo aver fatto ciò si può installare Libvips tramite:

```
tar xf vips-x.y.z.tar.gz
cd vips-x.y.z
./configure
```

Sul terminale, dopo l'ultimo comando verranno specificate eventuali dependencies da installare. E' consigliato l'utilizzo del comando `sudo apt-get update`. Per concludere, quando il comando `./configure` non crea problemi:

```
make
sudo make install
sudo ldconfig
```

4 Caratteristiche del linguaggio

4.1 Grammatica

4.1.1 Tipi

In seguito ad analisi relative ai possibili usi del linguaggio proposto, sono stati individuati i seguenti tipi primitivi:

- `integer`, utilizzato per rappresentare i numeri interi

Processor	Clock (GHz)	Cores	Real Time (s)	Speedup
Ryzen Threadripper 3970x	3.7	32 (64ht)	0.2	10 x
E5-2630V3 dual	2.4	2x8 (32ht)	0.44	8.4 x
Ryzen 7 3700X	3.6	8 (16ht)	0.44	5.6
AMD EPYC 7401P	2	24	0.48	10.2 x
I9-9900K	5	8 (16ht)	0.55	4.4 x
E5-2695V3 dual	2.3	28 (56 ht)	0.56	10.4 x
I7-6700K	4.4	4 (8 ht)	0.7	4 x
I7-3930K	3.2	6 (12 ht)	0.79	5.9 x
dual E5649 6core	2.5	12 (24 ht)	0.80	10.8 x
E5-1650	3.2	6 (12 ht)	0.87	5.94 x
I7-6700	3.2	4 (8 ht)	0.89	3.3 x
I7-8550U (XPS laptop)	1.8	4 (8 ht)	0.91	3.2 x
Xeon X5560	2.8	8 (16 ht)	1.08	13 x
Itanium2	?	64	1.1 (est.)	39.4 x
Intel I7 MacBook Pro	2.6	4 (8HT)	1.31	3.7 x
I5-3470S (IMac 27")	2.9	4	1.47	3.1 x
Xeon E5402 (64 bit)	2.0	8	1.88	7.3 x
Opteron 8220 (64 bit)	3.0	8	1.96	7.6 x
Phenom II X6	3.2	6	2.39	4 x
I7-3540m laptop (64b)	3.0	2 (4ht)	2.58	2.2 x
Dual quad-core Intel (64 bit)	3.0	8	2.8	7 x
I5-3210M	2.5	2 (4 ht)	3.51	1.8 x
Core 2 Extreme Quad (32 bit)	2.66	4	3.69	3.8 x
I5-5200U	2.20	2 (4 ht)	3.75	1.9 x
Opteron 850 (HP server)	2.4	4	4.25	3.7 x
Core 2 Duo (MacBook)	2.26	2	5.81	1.9 x
Opteron 254 (HP workstation)	2.7	2	6.14	1.9 x
P4 Xeon (64 bit)	3.6	2 (4 ht)	7	2.4 x
Core Duo (IMac)	2.0	2	11.5	1.85 x
ARM A15 Exynos 5 Chromebook	1.7	2	12.6	1.7 x
P4 Xeon (32 bit)	3.0	2 (4 ht)	19.7	1.6 x
ARM Exynos 5420 (-O0)	1.8-9 or 1.3	8 (4 big + 4 little)	21.31	2.2x
ARM A7 quad core Raspberry PI 2 B (32 bit)	1	4	21.6	3.9 x
PM (HP laptop)	1.8	1	31.8	--
P4 (Dell desktop)	2.4	1	36.6	--
EeePC atom/ssd	1.6	1 (2 ht)	41.5	1.6 x

Figura 4: Risultato benchmark

- `doublePrecision`, utilizzato per rappresentare i numeri in virgola mobile
- `str`, utilizzato per rappresentare le stringhe di caratteri
- `img`, utilizzato per rappresentare le immagini

E' stato inoltre individuato il tipo composto:

- `list`, tipo ricorsivo

Le variabili di tipo *int*, *doublePrecision* e *str* vengono dichiarate senza specificare un tipo esplicito.

Esempio di dichiarazione ed assegnazione:

```
stringVariable="this is a string";
intVariable=5;
doubleVariable=5.5;
img imageVariable="/path/img.png";
list li={1,2,"element"};
```

V1\V2	int	double	string
int	+ - * / or and CMP	+ - * / or and CMP	*
double	+ - * / or and CMP	+ - / * CMP	+
string	+ *	+	+ CMP

Tabella 1: Operazioni consentite dal linguaggio, sono da intendersi come V1 <operatore> V2. Con CMP si indicano le operazioni di comparazione

4.1.2 Operazioni

Il linguaggio consente le seguenti operazioni : somma, sottrazione, moltiplicazione, divisione, and e or logici, e paragoni (==, !=, >, >=, <, <=).

- Il risultato di divisioni, prodotti, somme e differenze fra *integer* e *doublePrecision* è sempre un *doublePrecision*.
- Il risultato di divisioni fra *integer* è un *doublePrecision*
- Il risultato di somme, prodotti e differenze fra tipi uguali mantiene il tipo degli elementi iniziali

In generale, *print(var + string)* stamperà su schermo la concatenazione delle due variabili, indipendentemente dal tipo di *var* (che verrà convertita a stringa)

String1 == String2 valuta il contenuto delle stringhe
String1 != String2 valuta il contenuto delle stringhe
print(3*string) stamperà per tre volte il contenuto di string
print(3+string) stamperà la concatenazione fra la 3 e il contenuto di string
print(string1+string2) stamperà la concatenazione fra le due stringhe

Le combinazioni di tipi consentite per ogni operatore sono riassunte nella tabella 1. I paragoni e gli operatori logici restituiscono sempre degli int, tutti gli altri operatori restituiscono un valore del tipo della colonna, tranne la moltiplicazione `int * string` equivalente a `string * int`. Le operazioni illustrate non sono ammesse per il tipo immagine e per il tipo lista.

4.1.3 Strutture di controllo

Il linguaggio fornisce la possibilità di usare sia comandi condizionali che comandi iterativi. I comandi condizionali sono i seguenti:

- `if ( condition ) then { instructions };`
- `if ( condition ) then { instructions } else { instructions }`

I comandi iterativi sono qui elencati :

- `while ( condition ) do { instructions }`
- `foreach ( tempElement :  list ) { instructions }`

Nell'espressione `while` si itera indefinitivamente finché la condizione non è più verificata. Nle costrutto `foreach` si itera invece per ogni elemento contenuto nella lista. E' possibile effettuare operazioni sull'elemento della lista tempElement, anche nel caso in cui si tratti di un'immagine. Questo rende possibile l'elaborazione di una lista di immagini.

4.1.4 Funzioni

Il linguaggio presenta una ricca selezione di funzioni built-in. All'utente viene inoltre fornita la possibilità di definire la propria funzione custom, con sintassi:

"`def nome_funzione ( parametri_formali ) { istruzioni }`".

4.2 Parser

Il linguaggio è stato progettato con i tools Bison e Flex, due strumenti per costruire programmi che gestiscano input strutturati. Flex si occupa dell'analisi lessicale (lexing) mentre Bison si occupa dell'analisi sintattica (parsing). L'output dei due tools è un parser di tipo LALR(1) che effettua un parsing bottom-up, con cui è possibile gestire produzioni left-recursive, utilizzando Bison per generare il parser e Flex per riconoscere i token nella fase di lexing.

Riportiamo di seguito le regole di produzione in formato BNF del linguaggio proposto:

Si possono eliminare le ambiguità della grammatica attraverso strumenti che permettono di specificare la precedenza degli operatori eliminando le possibili ambiguità e scartando automaticamente gli alberi di parsing non validi, consentendo l'utilizzo di una grammatica più semplice e compatta. A tal proposito sono utilizzati gli operatori di precedenza

4.3 Lexer

Nel lexer vengono adoperate delle regex per matchare i nomi delle variabili,le stringe,i numeri interi,i numeri in virgola mobile, l'inserimento di commenti, i simboli per i confronti, operazioni consentite sui tipi, i simboli consentiti, le funzioni built-in del linguaggio e le strutture di controllo.

Tipi

```
/* string */
\"{1}.[^"]*\"{1}

/* int */
[0-9]+

/* double */
[0-9]+\.{1}[0-9]+
```

Nome variabili

```
/* names */
[a-zA-Z][a-zA-Z0-9]*
```

Simboli

```
"+"
"-"
"*"
"/"
"="
"|"
","
";"
"("
"{"
"}"
```

":"
")"