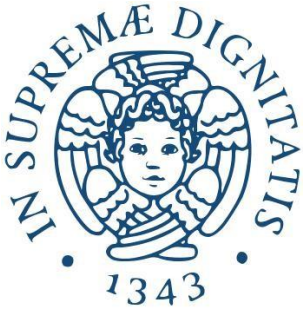
Università degli Studi di Pisa

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica



REPORT DI PROGETTO

ImgWatermarkPar: tool parallelo per il watermark di un set di immagini

Studente:
Antonio Sisbarra

Professore:
Prof. Marco Danelutto

Anno Accademico 2017/2018

1. Design

Le specifiche richiedevano l'implementazione di un programma parallelo per effettuare il watermark di un insieme di foto contenute in una directory.
Da una prima analisi del problema è risultato che leggere e scrivere immagini su disco richieda più tempo di calcolare il watermark (70% vs 30% circa del lavoro complessivo), di conseguenza sono stati pensati e sviluppati due modelli paralleli diversi per risolvere il problema.

Il primo modello viene chiamato da qui in poi "*SimpleFarm*", in quanto consiste sostanzialmente in una farm dove ogni lavoratore fa il seguente lavoro:

- *Legge* un'immagine da disco
- Effettua il *marking* dell'immagine
- *Scrive* l'immagine col mark su disco

Il secondo modello, da qui in poi chiamato "*FarmPipe*", rappresenta un'estensione del modello precedente, dato che ogni lavoratore all'interno della farm è composto da una pipeline di due elementi:

- Il primo *legge* un'immagine ed effettua il *marking* dell'immagine
- Il secondo *scrive* l'immagine col mark su disco

Di seguito vengono esposte le rappresentazioni grafiche dei due modelli implementati.

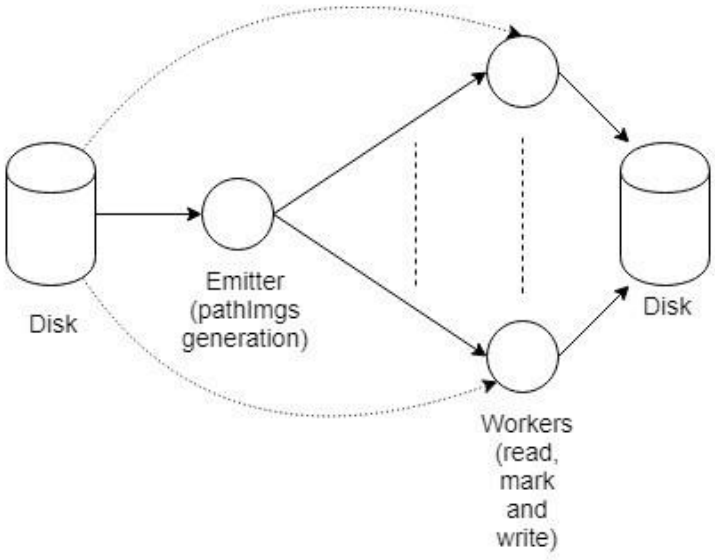


Figura 1 - Design SimpleFarm

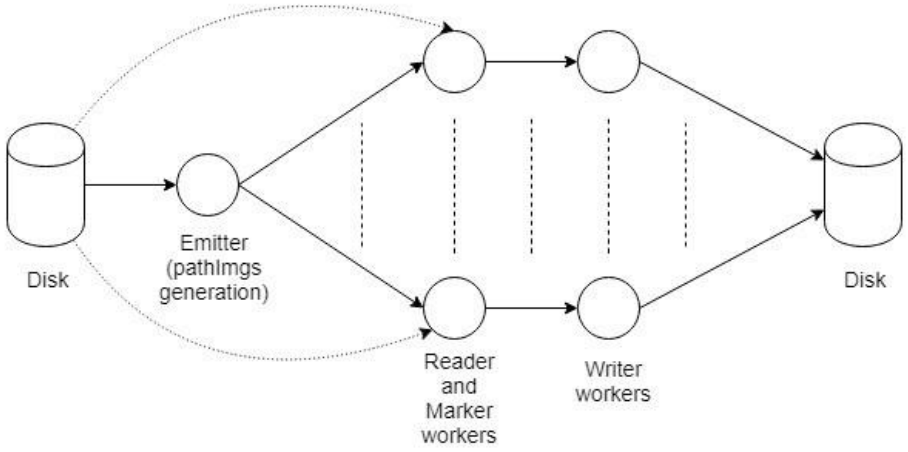


Figura 2 - Design FarmPipe

2. Performance modelling

2.1 Variabili utilizzate

Per costruire un modello di previsione delle performance sono state pensate 4 variabili di tempo per ciascun lavoro svolto durante l'esecuzione del programma:

- tGen = tempo necessario per generare il *path di una singola immagine*
- *t*Read = tempo necessario per leggere un'immagine in memoria
- tMark = tempo necessario per modificare i pixel corrispondenti al mark
- tWrite = tempo necessario per scrivere l'immagine col mark su disco

Per stimare queste sono stati svolti diversi test sulla macchina condivisa *XEON Phi* a disposizione degli studenti, utilizzando la media aritmetica dei tempi necessari alle singole fasi nell'implementazione sequenziale.
In seguito a questi test i valori stimati delle variabili sono risultati:

- tGen = 38 µsec
- *t*Read = 23.2 msec
- tMark = 36.1 msec
- tWrite = 34.1 msec

Dato che la fase di generazione dei path delle immagini richiede un tempo molto piccolo, questa varabile non viene considerata nella stima delle performance dei modelli.

2.2 SimpleFarm modelling

Data la natura del modello la stima (ideale) del tempo necessario per l'esecuzione (*completion time*) è data da:

$$\text{TSimpleFarm}(\text{Nimg}, \text{Nworkers}) = (\text{tRead} + \text{tMark} + \text{tWrite}) * (\frac{Nimg}{Nworkers}) + \text{tRead}$$
$$= (93.4\ \text{msec}) * (\frac{Nimg}{Nworkers}) + 23.2\ \text{msec}$$

2.3 FarmPipe modelling

La stima in questo caso è leggermente differente, nella pipeline infatti domina il tempo più alto per svolgere un lavoro. Inoltre, il numero di pipeline è la metà del numero di lavoratori, quindi si ha:

$$\text{TFarmPipe}(\text{Nimg}, \text{Nworkers}) = \max\{\text{tRead} + \text{tMark}, \text{tWrite}\} * (\frac{Nimg * 2}{Nworkers}) + \text{tRead}$$
$$= (\text{tRead} + \text{tMark}) * 2 * (\frac{Nimg}{Nworkers}) + \text{tRead}$$
$$= (118.6\ \text{msec}) * (\frac{Nimg}{Nworkers}) + 23.2\ \text{msec}$$

Osservando queste stime verrebbe spontaneo preferire il primo modello, ma da come vedremo dagli esperimenti (par. 4) sono stati prodotti risultati non così banali.
Inoltre, è presente un tRead costante in entrambe le stime, conseguenza naturale del fatto che la lettura dell'immagine che contiene il mark non è parallelizzabile (es. di *serial fraction* della *Amhdal Law*).

3. Implementazione

3.1 Set di immagini

Sono state utilizzate circa 970 immagini provenienti da due siti web[1][2] differenti, ridimensionate tutte su una dimensione 1024x768.
Queste immagini sono contenute nella directory *img_inp1024x768*.

3.2 Direttive di compilazione

Per la compilazione dei file è stato utilizzato Makefile, di conseguenza l'unico comando da utilizzare è: *make*, nella directory dove sono contenuti i file sorgenti (implementazione sequenziale, 2 modelli utilizzando thread, implementazioni FastFlow).

[1] http://lear.inrialpes.fr/
[2] http://www.imageemotion.org/

Per il corretto funzionamento dell'applicazione è necessario che nel path */usr/local/include* sia presente la directory della libreria FastFlow e il file *CImg.h* (libreria CImg).

3.3 Direttive di esecuzione

L'eseguibile sequenziale (*imgWatermarkSeq*) ha bisogno di 2 o 3 parametri per lavorare:

- il nome del file che contiene il mark
- directory con le immagini di input (con o senza / alla fine del nome)
- directory dove inserire le immagini col mark (non obbligatorio)

Gli altri eseguibili oltre ai parametri sopra citati hanno anche bisogno del *parDegree*, un intero che in *SimpleFarm* rappresenta il numero di lavoratori della farm, in *FarmPipe* rappresenta il numero complessivo di worker tra pipe e farm.

```
[spm18-sisbarra@C6320p-2 ImgWatermark]$ ls
CImg.h                  imgWatermarkFarmPipe.cpp      imgWatermarkSimpleFarm.cpp
img_inp1024x768         imgWatermarkFFFarmPipe        Makefile
img_inp1024x768_POCHE   imgWatermarkFFFarmPipe.cpp    mark1024x768.jpg
img_inp640x480          imgWatermarkFFSimpleFarm      mark640x480.jpg
img_inp640x480_POCHE    imgWatermarkFFSimpleFarm.cpp  myqueue.cpp
img_out1024x768         imgWatermarkSeq               utils.cpp
img_out640x480          imgWatermarkSeq.cpp           watermarkUtil.cpp
imgWatermarkFarmPipe    imgWatermarkSimpleFarm
[spm18-sisbarra@C6320p-2 ImgWatermark]$ ./imgWatermarkSimpleFarm 64 mark1024x768
.jpg img_inp1024x768/ img_out1024x768/
dirInput is: /home/spm18-sisbarra/ImgWatermarkProject/ImgWatermark/img_inp1024x7
68/
dirOutput is: /home/spm18-sisbarra/ImgWatermarkProject/ImgWatermark/img_out1024x
768/
Reading markimg file...
Reading markimg ok, now starting reading images...
Working on imgs...
----------------------------
- Computed 976 imgs marking using 64 threads in 3165 msecs
----------------------------
[spm18-sisbarra@C6320p-2 ImgWatermark]$
```

Figura 3 - Esempio di esecuzione con parametri settati correttamente

3.4 Architettura e strutture dati

Ogni worker della farm è rappresentato da un thread che, nel caso della *SimpleFarm* svolge tutto il lavoro di lettura, mark e scrittura, mentre nel caso della *FarmPipe* crea a sua volta un thread che rappresenta il secondo stage della pipeline, per ogni worker.

Per la comunicazione tra i vari stage viene utilizzata essenzialmente una *deque*, incapsulata dalla classe *myqueue*. La coda consente di aspettare un nuovo task in caso non ce ne siano utilizzando una variabile condizione.

La generazione e lo smistamento dei task viene svolto dal thread principale (l'esecutore del main()), utilizzando una politica di tipo Round Robin con un indice circolare tra i vari thread worker. Questa scelta è sembrata abbastanza appropriata dato che tutte le immagini hanno dimensione uguale e lavorare sui pixel porta via lo stesso tempo di lavoro.

3.5 Strutture FastFlow

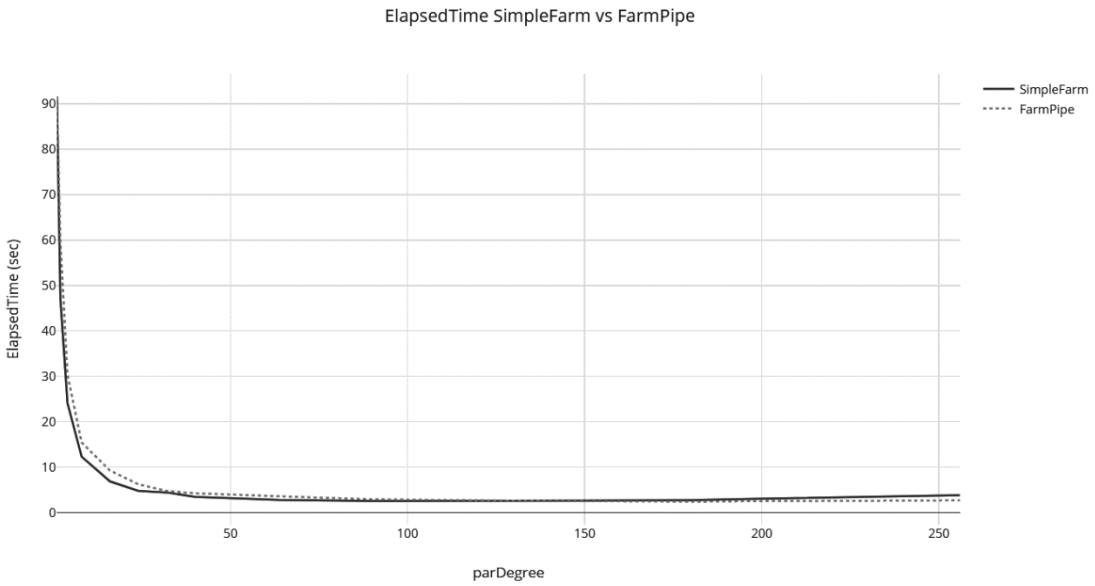
Nel progetto sono presenti le implementazioni FastFlow dei due modelli proposti, *FFSimpleFarm* e *FFFarmPipe.*
Nel primo caso viene utilizzata la struttura *ff_Farm* con un emitter costituito dalla generazione dei path, mentre nel secondo caso viene utilizzata una *ff_Pipe* annidata nella farm.
Utilizzando FastFlow non vi è più il bisogno di utilizzare una coda, visto che lo scambio di dati viene offerto dalla libreria.
Lo scheduling utilizzato nella farm è on_demand.

4. Esperimenti e risultati comparativi

In questo capitolo verranno mostrati diversi risultati ottenuti sulla macchina condivisa *XEON Phi,* utilizzando un dataset di 976 immagini. Innanzitutto, viene mostrato un confronto tra i due modelli pensati per capire quale sia il modello più adatto ai nostri bisogni, verrà poi effettuato un confronto tra le performance reali sulla macchina e quelle prevista dal modello delle performance (capitolo 2). Infine, verrà mostrato un confronto tra le performance dei modelli sviluppati senza framework e le performance avute con l'ausilio di FastFlow.

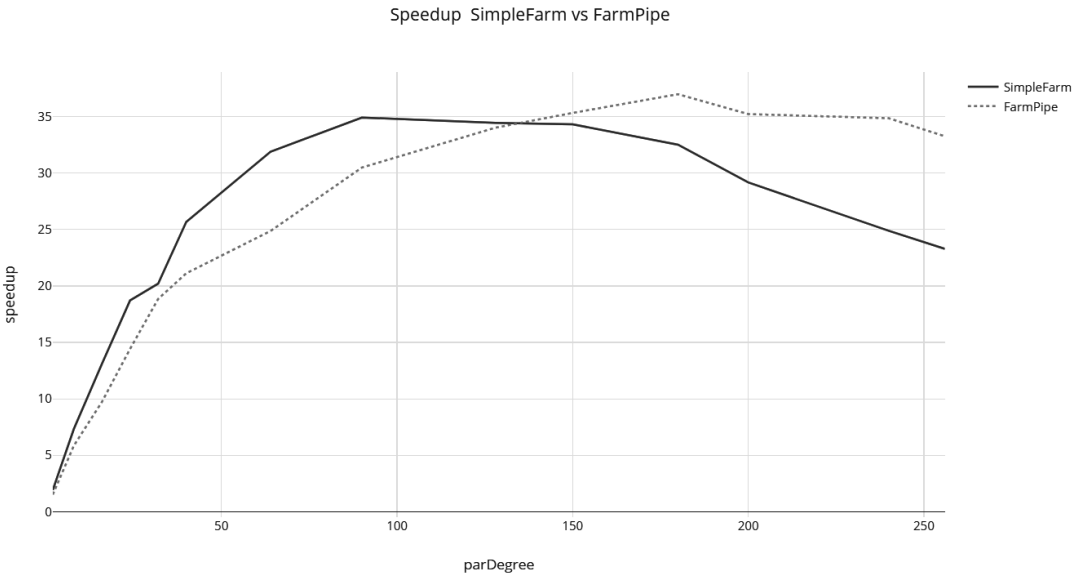
4.1 Confronti SimpleFarm FarmPipe

Il primo metro di paragone utilizzato per confrontare i due modelli è il tempo di esecuzione complessivo dell'applicazione (*latency*).



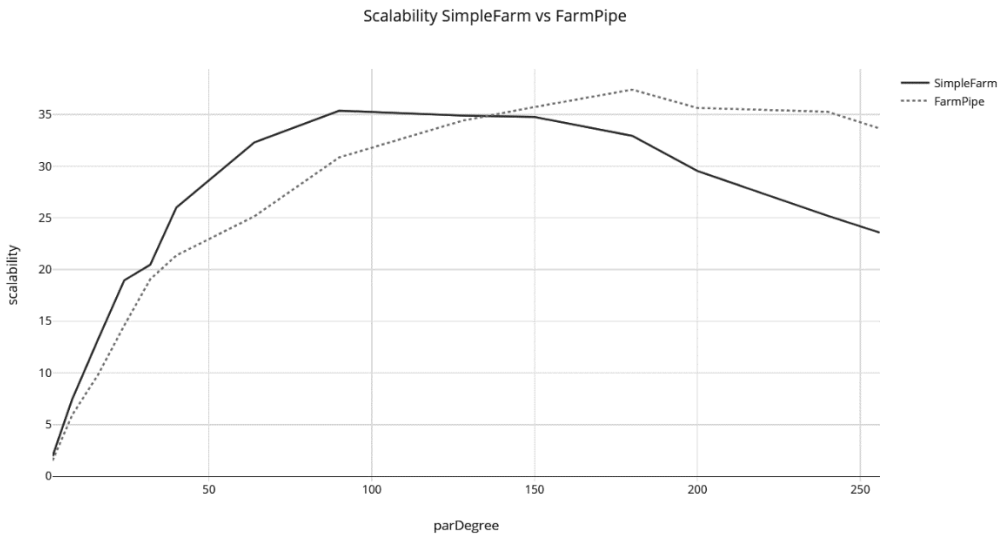
Dal grafico si può notare a grandi linee un andamento molto simile delle due funzioni, con SimpleFarm che raggiunge prima di FarmPipe il suo minimo (2.59 sec). Con un numero di thread molto alto (da 180 circa in poi) il modello FarmPipe si comporta leggermente meglio.

Il secondo metro di paragone tra i due modelli è l'andamento dello *speedup,* espressa come rapporto tra tempo sequenziale e tempo parallelo con numero variabile di worker.



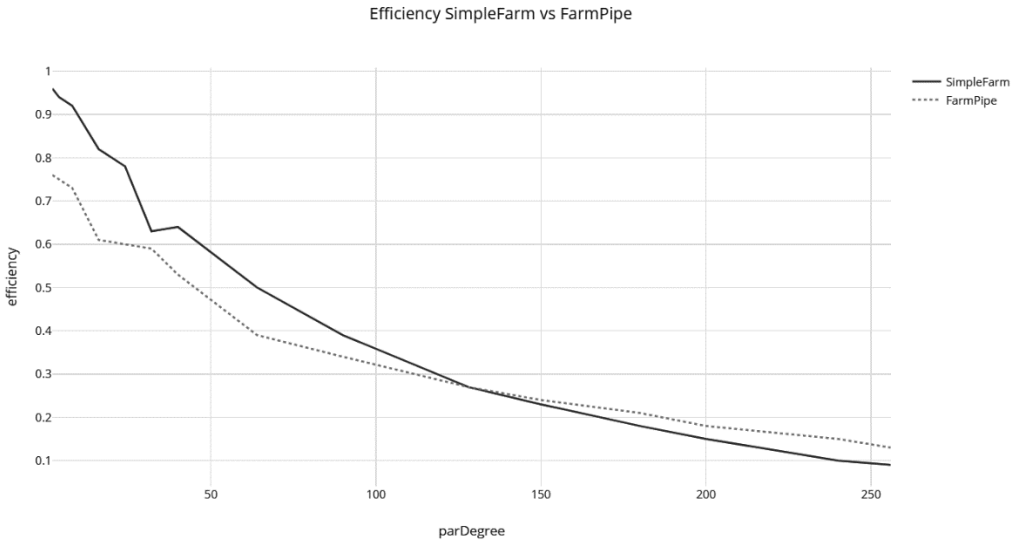
Si nota subito che lo *speedup* di SimpleFarm comincia ad essere decrescente prima di FarmPipe, anche se quest'ultimo fino a circa 125 thread offre un guadagno minore in termini di prestazioni.

Il terzo metro di paragone è la *scalability* espressa come rapporto tra tempo parallelo di un worker e tempo parallelo con numero variabile di worker.



Il grafico rimane sostanzialmente identico, conseguenza del fatto che il tempo parallelo con un worker è praticamente identico al tempo sequenziale (91 sec vs 90 sec circa).

L'ultimo metro di paragone è l'*efficiency* espressa come rapporto tra *speedup* e numero di worker utilizzati. Questo ci consente di capire quanto effettivamente è utile aumentare il numero di worker a disposizione.

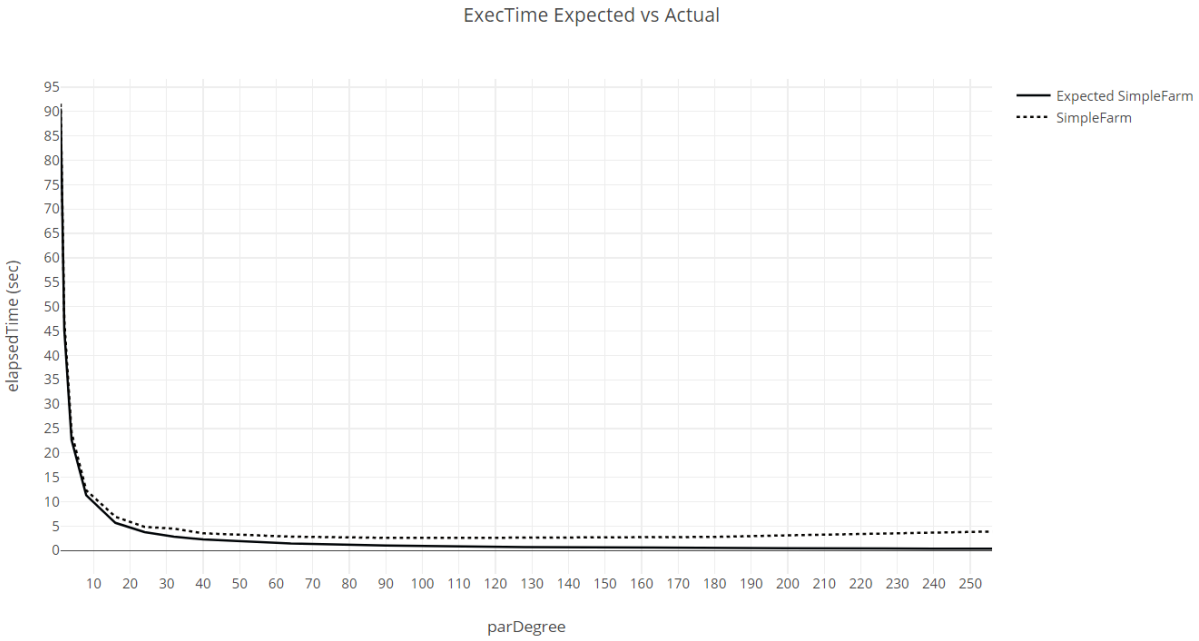


L'efficiency comincia ad essere bassa da circa 40-45 worker per entrambi i modelli, con SimpleFarm che domina su FarmPipe fino a 125 worker circa.

Da questi confronti quindi si può dedurre che il modello SimpleFarm si comporta meglio del FarmPipe fino ad un numero consistente di worker (sui 150), poi il FarmPipe sfrutta leggermente meglio le risorse a disposizione.

4.2 Confronti con modello delle performance

Nello studio delle misure di performance è importante capire quanto ci si discosta da quelle che sono le stime teoriche e ideali fatte (cap.2). Di seguito viene fatto il confronto tra i tempi di esecuzione stimati e reali.

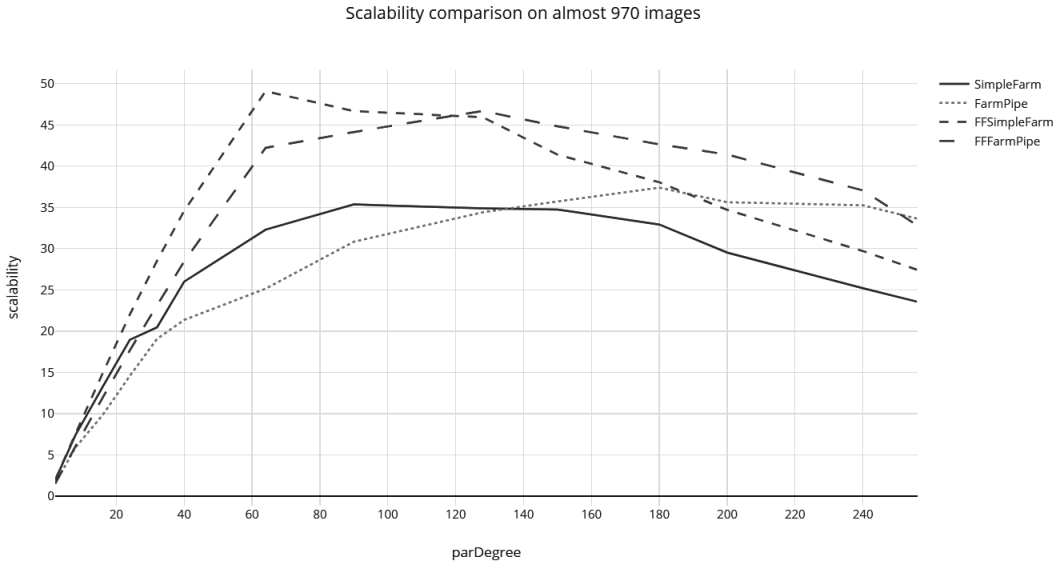


Dal grafico si evince che fino a circa 2 worker la differenza di prestazioni tra stima e realtà è molto sottile. Da lì in poi il tempo di esecuzione della stima tende a 0, mentre il tempo reale tende a stabilizzarsi e a non decrescere più, anzi da 120 worker il tempo d'esecuzione tende a salire.
Questo comportamento dell'applicazione è dovuto principalmente al limite imposto dalle operazioni di I/O con la memoria secondaria e, inoltre, al crescere dei worker cresce l'overhead di pari passo.

4.3 Confronti con implementazioni FastFlow

Segue un confronto tra i modelli implementati non utilizzando framework e le implementazioni con FastFlow.


Speedup comparison on almost 970 images
SimpleFarm
FarmPipe
FFSimpleFarm
FFFarmPipe
speedup
parDegree
0
5
10
15
20
25
30
35
40
45
50
20
40
60
80
100
120
140
160
180
200
220
240


Scalability comparison on almost 970 images
SimpleFarm
FarmPipe
FFSimpleFarm
FFFarmPipe
scalability
parDegree
0
5
10
15
20
25
30
35
40
45
50
20
40
60
80
100
120
140
160
180
200
220
240


Efficiency comparison on almost 970 images
SimpleFarm
FarmPipe
FFSimpleFarm
FFFarmPipe
efficiency
parDegree
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
20
40
60
80
100
120
140
160
180
200
220
240

Dai tempi di esecuzione non risulta un grosso miglioramento in termini assoluti delle performance, anche se a lungo andare ci sono piccoli miglioramenti, di circa mezzo secondo sul minimo.
La differenza sostanziale con FastFlow sta in un miglior *speedup* e *scalability*, specialmente dai 40 worker in su.
Anche i valori di *efficiency* risultano migliori, in quanto il calo brusco avviene dopo (intorno ai 70 worker) rispetto all'implementazione standard (dai 30 worker in poi)
Inoltre, anche con FastFlow il modello SimpleFarm risulta migliore rispetto al modello FarmPipe fino ai 125 worker circa.

5. Conclusioni

Giunti alla fine di questo report risulta chiaro che il modello SimpleFarm ha prestazioni leggermente migliori rispetto a FarmPipe, utilizzando un numero di worker non superiore ai 125 circa.
Si può senz'altro dire che l'applicazione funziona discretamente su un'architettura multi-core, come la XEON Phi, consentendo il *marking* di quasi 1000 immagini di dimensioni medio-grandi (1024x768) in due secondi e mezzo circa. Risultato sicuramente ottimo se paragonato alla versione sequenziale che richiede come visto un minuto e mezzo circa sullo stesso dataset di immagini.
Un'altra versione possibile avrebbe potuto rendere parallela anche la fase di marking delle immagini, essendo un problema *embarassingly data parallel,* cioè con una completa indipendenza tra i pixel di una stessa foto. Questa possibilità è stata scartata dall'inizio, a causa dell'overhead richiesto a fronte del poco guadagno sulla computazione.
Dall'implementazione FastFlow inoltre si è visto un leggero miglioramento al crescere del numero di worker richiesti, ed un minor numero di righe di codice e di complessità di sviluppo richiesta.