**Raspberry Pi Phash Cluster**

**Speedup?**

**Jamar Mitchell**(*Author*)

UOttawa Faculty of Engineering Student

Prof. M. Bolic

Ottawa, Canada

jmitc008@uottawa.ca

**Elijah Balo** (*Author*)

UOttawa Faculty of Engineering

Prof. M. Bolic

Ottawa, Canada

ebalo061@uottawa.ca

***Abstract*—In computer architecture, Amdahl's law (or Amdahl's argument) is a formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.The document conducts an experiment on amdahl's law with perceptual hashing.**

**Keywords— *RPI; SpeedUp:Phash***

**I. Introduction**

Hashing is a function that takes in input and maps them to strings/numbers   
The function is called the hash function  
The mapped result is also known as the message digest.

Similarly to hashing there is a method called phashing. The use of the Phash function is to generate a message digest(fingerprint) of a multimedia file.  
Unlike the regular cryptographic hashing that generates drastic changes in output based on a small change in the input,Phash outputs are similar if the features of the of the inputs are also similar.  
.The purpose of this report is to get a list of images and calculates their perceptual hashes. The images will be distorted by rotating them and using the phash function to recalculate their hashes.  
The task of hashing all the images in a list would be carried out by one worker thread (raspberry pi) and the execution time for each test would be measured and displayed in the output.  
The same task is then split across two worker threads and the new execution time is the again measured and displayed in the output.  
The two measured execution times for the single raspberry pi and the raspberry pi clusters will then be compared with each other and speed up will be calculated using Amdahl’s law to determine if it is faster to perceptual hash on two raspberry pis rather than one.

**A. Abbreviations and Acronyms**

**PHash**: Perceptual Hashing

**PIL**: Pillow

**RPI**: Raspberry Pi

**II.How Phash works**

The images are stored in a referenced data folder. The referenced image is then made converted into a hash value. To make a comparison to derive whether images are similar the image to compare is then slightly rotated or distorted and also given a hash value. If the distance ( the difference of the images ) calculated is a small value it is clear that the images are very similar. The lower the distance, the greater the similarity between the two images.

## **A. Compiling a *Python* program**

Assuming the *source code*  is saved as ‘*main.py*’. The following command is used to compile the python file and store it in an *executable file* *-main.*

**$ python -m py\_compile main.py**

The following command is used to run the python program:

**$ python main.py**

**III. Project Plan/ Process**

## **A. Phase 1 (Test 1)**

The plan in this test is to use the hash function to calculate a perceptual hash of the original image after which the image gets distorted by rotating it an angle of

**1 ° (degree)** and the new perceptual hash of the image is then calculated using the same hashing function. Being perceptual hashing, it is expected that the hash distance (difference between the original hash and the distorted hash) should vary but not by a vast amount as the distortion was minimal.

## **B. Phase 2 (Test 2)**

In this phase of the project, the original perceptual hash of the image is calculated before the image gets distorted by rotating an angle of 90e after which the image gets distorted by rotating it an angle of

**90° (degree)** and the new perceptual hash of the image is then calculated using the same hashing function. The perceptual hash distance is calculated and is expected to yield a higher value as the image was distorted by a vast degree. The values of the distances of the resulting hash is then compared to the values that was gotten in the first phase of this project to ensure consistency and to make sure that the algorithms function properly and efficiently.

**C. Phase 3 (Speed Up)**

In this section, the execution time of test1 and test2 was measured on a single raspberry pi. However, the goal of this phase is to setup a raspberry pi cluster of two pis and which would have double the processing power of just one pi. The two tests are then repeated on the raspberry pi cluster and the new execution time is calculated.

The speedup is calculated based on the execution time of the weaker processor vs the execution time on the enhanced processor. In theory, the execution time of the entire program should be doubled and the calculated speed-Up should yield a value of 50%.

The value gotten from the experiment is then compared with the theoretical calculation of Amdahl’s law..

# **IV. RESULTS**

The tables below, show the results we have received for the execution time for when we ran each task separately and together on a single RPI as well as running it parallel on two RPI. There is a speed up observed due to the task being run at the simultaneously in parallel rather than running one after another on a single RPI.

Table 1: Execution times in seconds for each test

|  |  |  |
| --- | --- | --- |
| Test | Single RPI (sec) | 2 RPI  (sec) |
| Test 1 | 43.327 | n/a |
| Test 2 | 51.451 | n/a |
| Test 1 and 2 | 69.37 | 53.147 |

TABLE 2: Speed up in percentage [(old-new/old)\*100]

|  |  |
| --- | --- |
| Speed Up (experimented) | Speed Up (calculated) |
| 23.14% | 50% |

**V. Conclusion**

The main goal of this project is to show and prove that using a cluster of Raspberry Pis rather than a single one to perform perceptual hashing would be faster in performance having smaller execution times and a speedup.

We implemented two tasks using parallel processing for the two Raspberry Pis to split and execute simultaneously. The two task entailed comparing the two calculated hashes by rotating an image by 1 degree while the second task involved comparing the two rotating the image by 90 degrees. Splitting the original check\_hash\_algotrithm() method into these two tasks and running them in parallel rather than on a single Pi was able to produce speed ups of around 24%.

Although the theoretical speedup was calculated to be 50% we expect that due to the small sample of 30 images we were able to use without the Raspberry Pis crashing we weren't able to produce a more reliable output and speedup. With that being said we believe that were we to have a considerable sample that we planned to have of around 1000 images the experimental speed up would have been vastly different and a lot closer to the theoretical speedup.

##### 

##### 

##### 

##### 

##### 

##### **References**

*Imagehash4.0.. Retrieved December 12,2017,from*[*https://pypi.python.org/pypi/ImageHash*](https://pypi.python.org/pypi/ImageHash)

Zhang, T., Kang, S., & Lipsky, L. R. (1995). *On the performance of parallel computers: order statistics and Amdahls Law*. Munich: Mathematisches Institut und Institut für Informatik der Technischen Universität München.

G., & I. (2017, November 21)*. How to Make a Raspberry Pi SuperComputer! Retrieved December12,2017,from*http://www.instructables.com/id/How-to-Make-a-Raspberry-Pi-SuperComputer/