Doppelgänger: a cache for approximate computing

Akshay Gadre(CS12B034)

1 Abstract

Conventional caches use a one-one mapping of tag array and data array indices to map an input address onto a data. The address is divided into three fields- Tag bits, Set Index Bits and Offset Bits. Doppelgänger[1] brings a novel approach to reduce the area and energy usage of LLCs by introducing tolerance in the accuracy of the data made available. It aims to reduce the amount of data that needs to be stored by trading off the accuracy of data made available to the user. Doppelgänger also utilises the similarity amongst the cache data blocks which is rampant in image processing and video processing algorithms making it an ideal choice for those scenarios.

2 Introduction

Doppelgänger is an approximation cache which trades off the area and energy usage of LLCs to the accuracy of the data supplied by them. Contradictory to the in-built guarantee of data accuracy by conventional caches, Doppelgänger will give inaccurate data but at a much reduced cost factor. This is thus a value based optimization versus reference based optimization where we are actually trading off the reliability with the actual footprint every memory transaction will have.

It aims to utilize the similarity amongst the cache data blocks to reduce the data needed to be stored in it. It proposes a novel many-one tag array to data array mapping architecture where each tag maps to one data array element while one data element can be mapped to by multiple tag array elements. However to provide all the above benefits, it has lose the accuracy of the actual data being supplied.

In this report, we will confine our discussion to image processing applications. In section 3, we discuss the architecture of the Doppelgänger cache along with description of basic read and write operations. We will then discuss the tools that were developed during the course of this project and discuss th e simulation criteria for which the simulation were run in section 4. We then describe the creation and properties of the test data in brief to emphasize how the selection of data was done to cover all the possible input sets in section 5. In the following section 6, we discuss the important results that the experiments generated, their analysis and implications for the use of this cache in actual applications. To conclude, in section 7 we wrap up the discussion with suggestions on how this project can be improved upon.

3 Architecture of the proposed cache

3.1 Tag Array

The tag array in the proposed architecture is that the set-indexing lines will remain the same. As we already know we will have multiple elements of the tag array pointing to the same element in the data array. The same is visible in figure 1. Each of the tag entries will however have a few more elements in them.

- **Address Tag** - This is the convention address that is usually used to compare the incoming address bits
- **Line's State** - This is decided upon by the set-indexing bits actually are making this element active
- **Prev and Next Tag Pointers** - These are the pointers to the next and previous tags which are pointing to the same mapped data element
- **Map value** - This is the mTag value that this element is pointing to in the data array of the cache

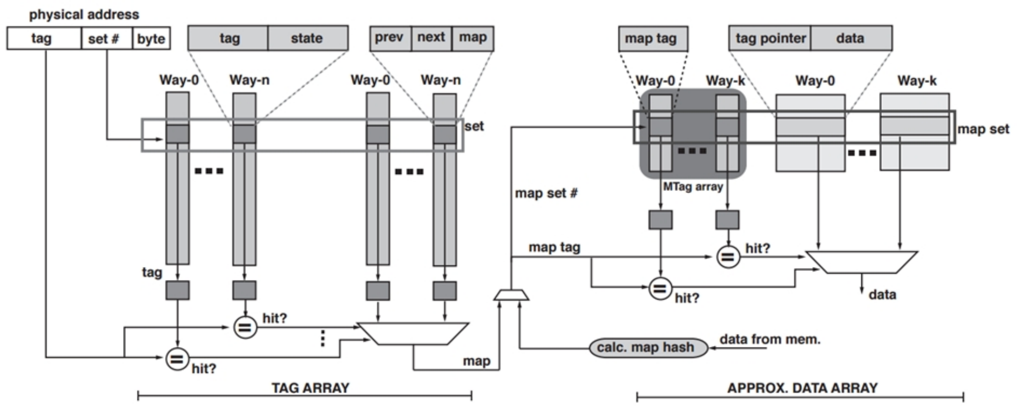


Figure 1: The Doppelgänger cache architecture[1]

3.2 Data Array

The data array in the proposed architecture is that the set-indexing lines will remain the same. As we already know we will have an entry of the data array are pointed to by multiple entries in the tag array. The same is visible in figure 1. Each of the data entries will however have a few more elements in them.

- **Map Tag** - Unlike conventional caches a data entry is not addressed by its address but by a map created from its data content. This is called the mTag.
- **Tag Pointer** - This is a pointer to one of the tag array elements pointing towards it.
- **Data block** - This actually contains the approximate data that is contained in the cache mapped to this map at the given point in time.

The mapping has been described comparitively in figure 2

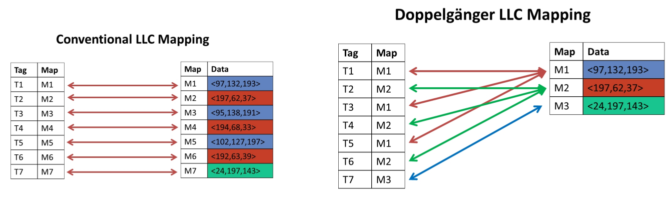


Figure 2: Mapping of tag array and data array for normal and Dopplegänger caches

3.3 Servicing Read requests from the cores

The algorithm to service read requests has been described in figure 3

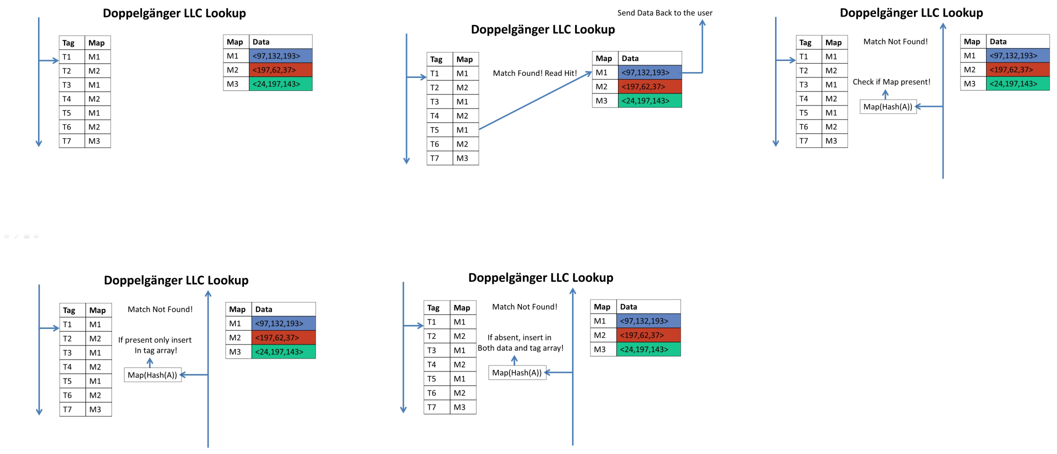


Figure 3: Servicing Read Requests

3.4 Servicing Write requests from the cores

The algorithm to service write requests has been described in figure 4

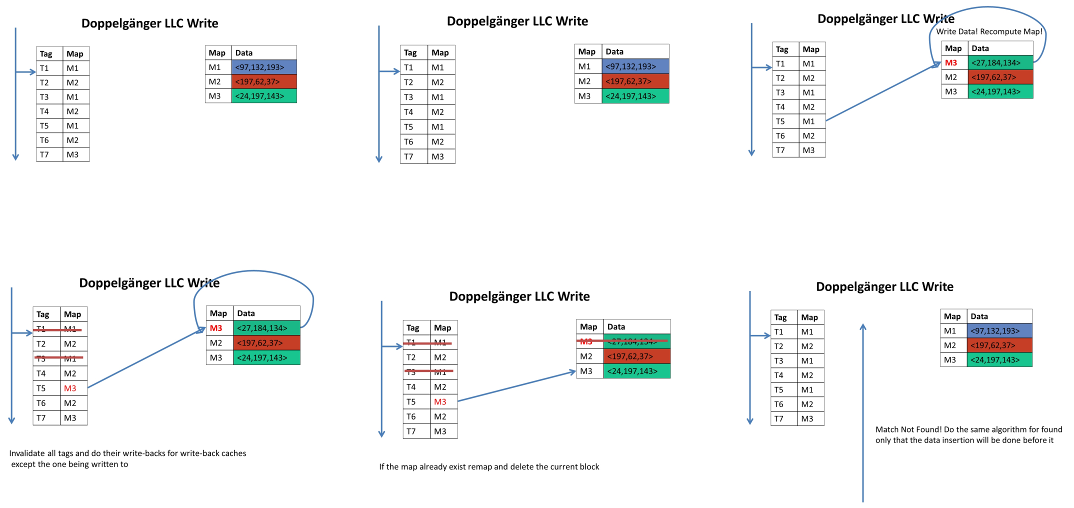

Doppelgänger LLC Write
Write Data! Recompute Map!
Invalidate all tags and do their write-backs for write-back caches except the one being written to
If the map already exist remap and delete the current block
Match Not Found! Do the same algorithm for found only that the data insertion will be done before it

Figure 4: Servicing Write Requests

3.5 Map function calculation

The algorithm to calculate the map function of a block has been described in figure 5. As is evident there are two parts to calculate the map:

3.5.1 Hashing function

This function is responsible for representing a whole host of pixels in a block as one pixel. This is can be done by many algorithms some of which are described later in the report and were tested with for the project

3.5.2 Mapping function

This function is responsible for further exponentiating the data error by reducing the pixel to a set of k-bit value which can be used as mTag value for the data block. This is can be done by many algorithms some of which are described later in the report and were tested with for the project

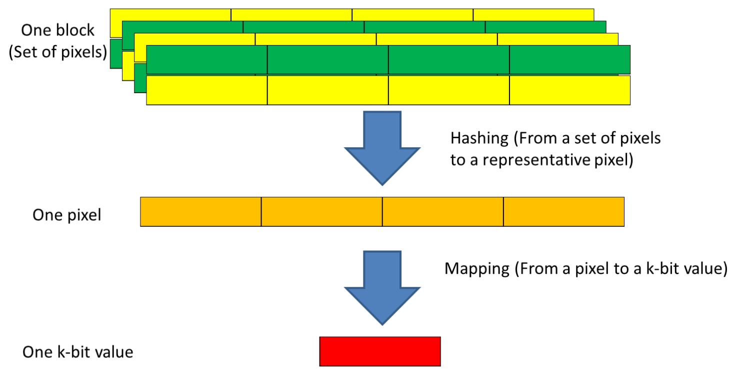

One block
(Set of pixels)
Hashing (From a set of pixels to a representative pixel)
One pixel
Mapping (From a pixel to a k-bit value)
One k-bit value

Figure 5: The mapping algorithm

4 Developed Tools and Simulation Criterion

One of the most important part of the project was to develop simulation tools for approximation caches and generate transaction traces suitable for the simulation tools from the images chosen for comparison. Upon receiving output traces, there was a need of extra tools for comparison and analysis of these traces to compute the cumulative read error and the no. of writes to the main memory by each. To do this, we even had to develop simulation tools for normal caches and generate their output traces for comparison.

In this section we will briefly describe all the tools and open functionalities that were developed.

4.1 Transaction Trace Generator

The transaction trace generator is pre-trained to generate 5 state-of-the-art transactions that typically occur in image processing applications which are:

1. Greyscaling - Where we read all the channels and put their average in all the three channels back

2. Blurring - Where we read all our neighbouring bits and become their average

3. Darkening - Where we read each pixel and halfen all of its channel values to darken it.

4. Blocking - Where we traverse the image in a blockwise fashion.
5. Channel Acquiring - Where we just set the bytes of all channels except the one chosen to 0.

For each image which it is given input, it will precompute all the write values which are corresponding to all writes and then it will generate a transaction trace per image per traversal type which will act as an input to the cache.

4.2 Normal Cache Simulator

The normal cache simulator takes in the parameters of the cache and the original main memory state and simulates the transaction traces and outputs the output traces.

4.3 Approximation Cache Simulator

The approximation cache simulator takes in the parameters of the cache and the original main memory state and simulates the transaction traces and outputs the output traces.

4.4 Cumulative Read Error Finder

Given two output trace files it comuptes the no. of writes to the main memory by each of the traces and compares their reads to get the cumulative read error which is given by the formula

$$CER = \sum_{k=1}^{numReads} |V_{d,k} - V_{n,k}|$$

where $V_{d,k}$ and $V_{n,k}$ are the two read values distorted and normal respectively.

4.5 Parameters that were considered and varied

4.5.1 Block Size

As is true with normal caches, the block size is even more important here since it not only influences the no. of tag and setIndex bits but is also the degree of information loss in the hashing phase for the approximation caches here. The cache sizes considered were 8 bytes, 16 bytes and 32 bytes as shown in figure 6.

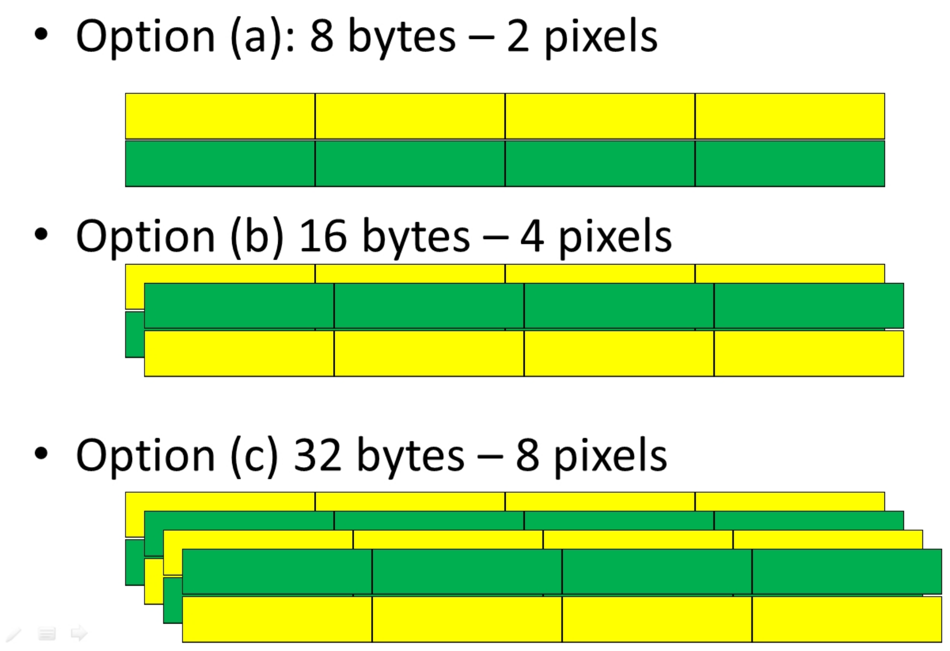

• Option (a): 8 bytes – 2 pixels
• Option (b) 16 bytes – 4 pixels
• Option (c) 32 bytes – 8 pixels

Figure 6: Block sizes that were considered

4.5.2 Hashing functions

The importance of the choice of hashing functions has been made evident in the previous section. We try out the following hashing functions:

- **Option (a)**: For each of the channel we take the average of the values of that channel of all pixels in the block
- **Option (b)**: For each of the channel we take the maximum of the values of that channel of all pixels in the block
- **Option (c)**: For each of the channel we take the minimum of the values of that channel of all pixels in the block

4.5.3 Mapping functions

The importance of the choice of mapping functions has been made evident in the previous section. The best results will intuitively be when we take the top-n bits of all channels in the mapping criterion. We try out the following mapping functions:

- **Option(a)**: n=2 ; 75% reduction in the size of tag
- **Option(b)**: n=4 ; 50% reduction in the size of tag
- **Option(c)**: n=6 ; 25% reduction in the size of tag
- **Option(d)**: n=8 ; 0% reduction in the size of tag

4.6 Simulation Parameters that were fixed

The simulation environment was prepared to work for 512px×512px and 24 bits per pixel images. Thus the parameters that were fixed were:

1. Associativity = 8
2. Physical Address Space = 20 bits → 1MB
3. Cache Size = 64 KB

5 Test Data

Each of the test images were chosen to bring out one of the specific attributes described above. Each image is 512px×512px and 24 bits per pixel.

The first image, in figure 7, which was chosen was that of a checkered board. It was chosen to test the intuitive notion that checkered boards have a a lot of similarity and this an approximation cache like Dopplegänger whould perform quite well for this image. Of course, this is an artificially manufactured image and hence a good starting point to verify our hypotheses

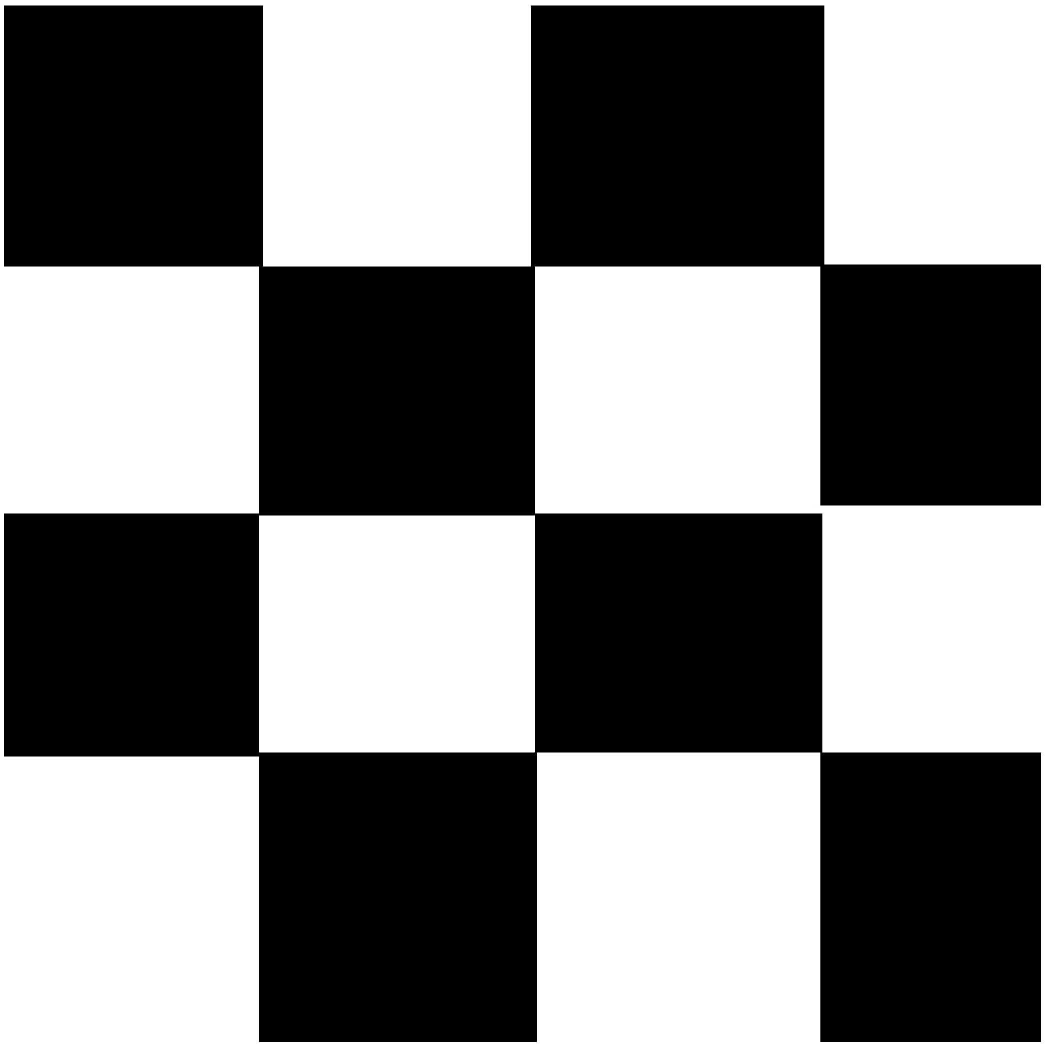


Figure 7: Arificial Image which is best scenario for Dopplegänger

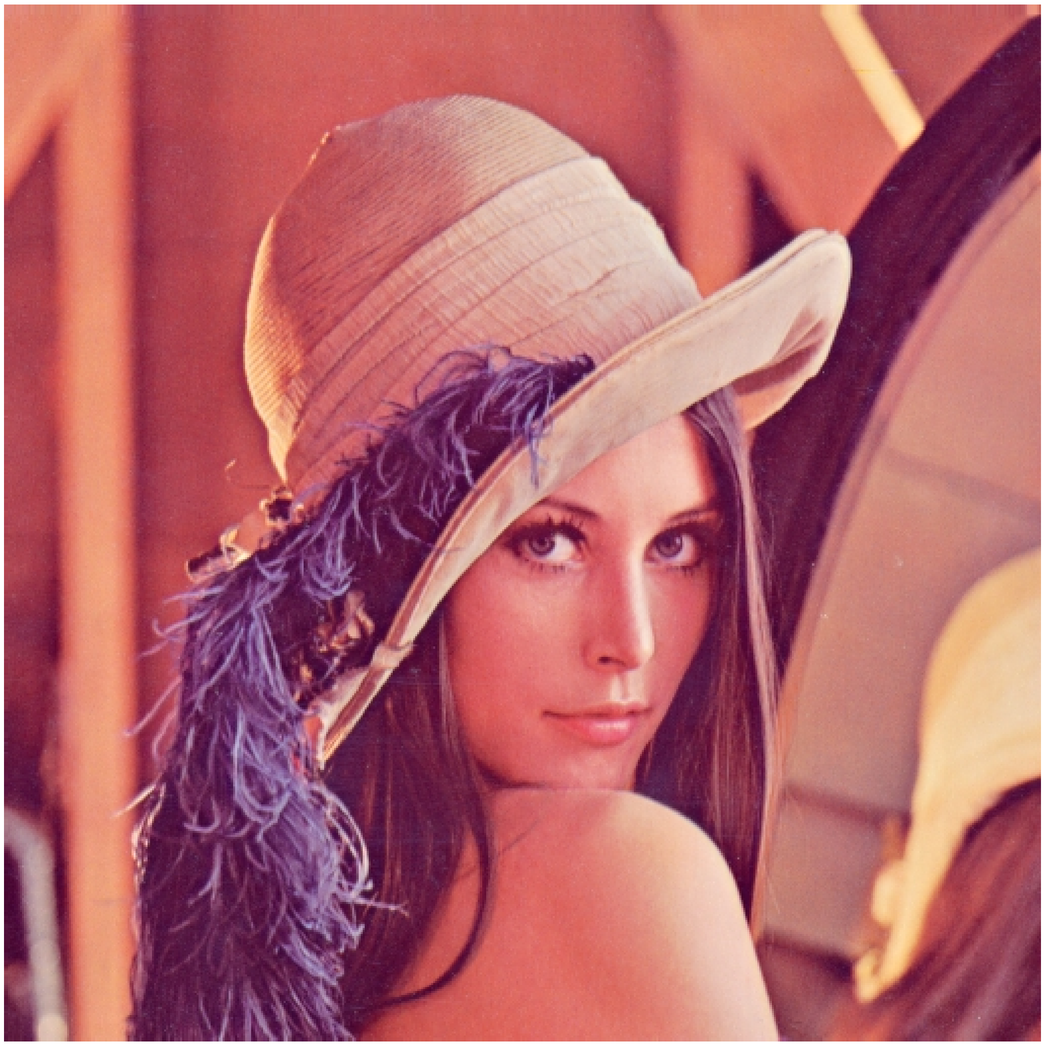


Figure 8: The most famous image processing image Lena

This second image, in figure 8, needs no further introduction to any image processing rookie in the world. This image is used more than "Hello world" is used while teaching programming in the computer vision world. This image also acts like one of the worst case scenarios for our approximation cache with nearly no similar areas and highly contrasting surfaces touching each other.

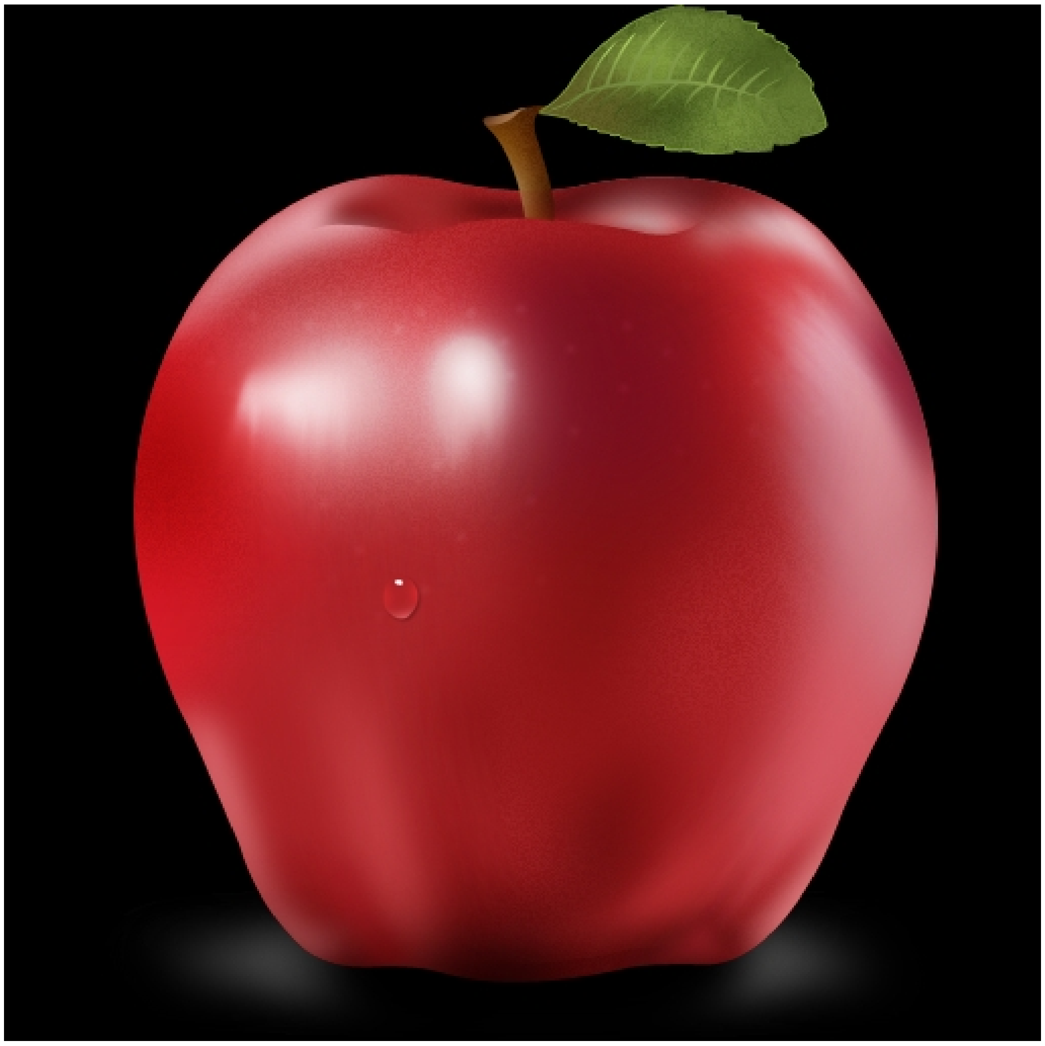


Figure 9: Real world image which is best case for Dopplegänger

The third image, in figure 9, was chosen because this is a real world example of similarity in a image. The red foreground and the black backgrounds gives our cache multiple chances to cache in on these similarity features. Note that this is a dark image which means the hash function chosen comes into play.

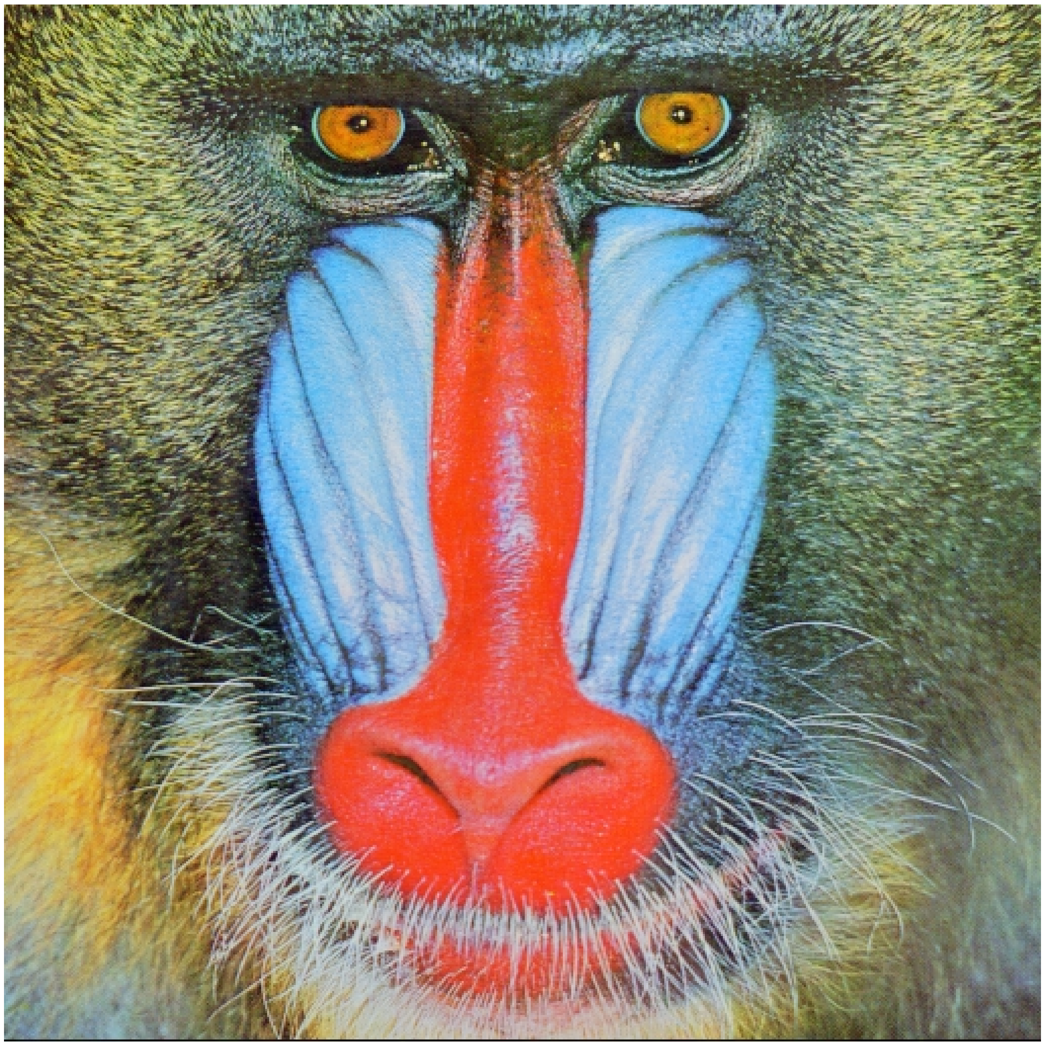


Figure 10: Baboon image where there is stark contrast in the values of each channel

The fourth image, in figure 10, was chosen to show how in a real world image the maximum of a channel can rule an area in an image like the red in the nose or the blue in the whiskers and the cheek. This is a good chance for our maximum hashing function to work better than the average since on one of the channels rules the each area.

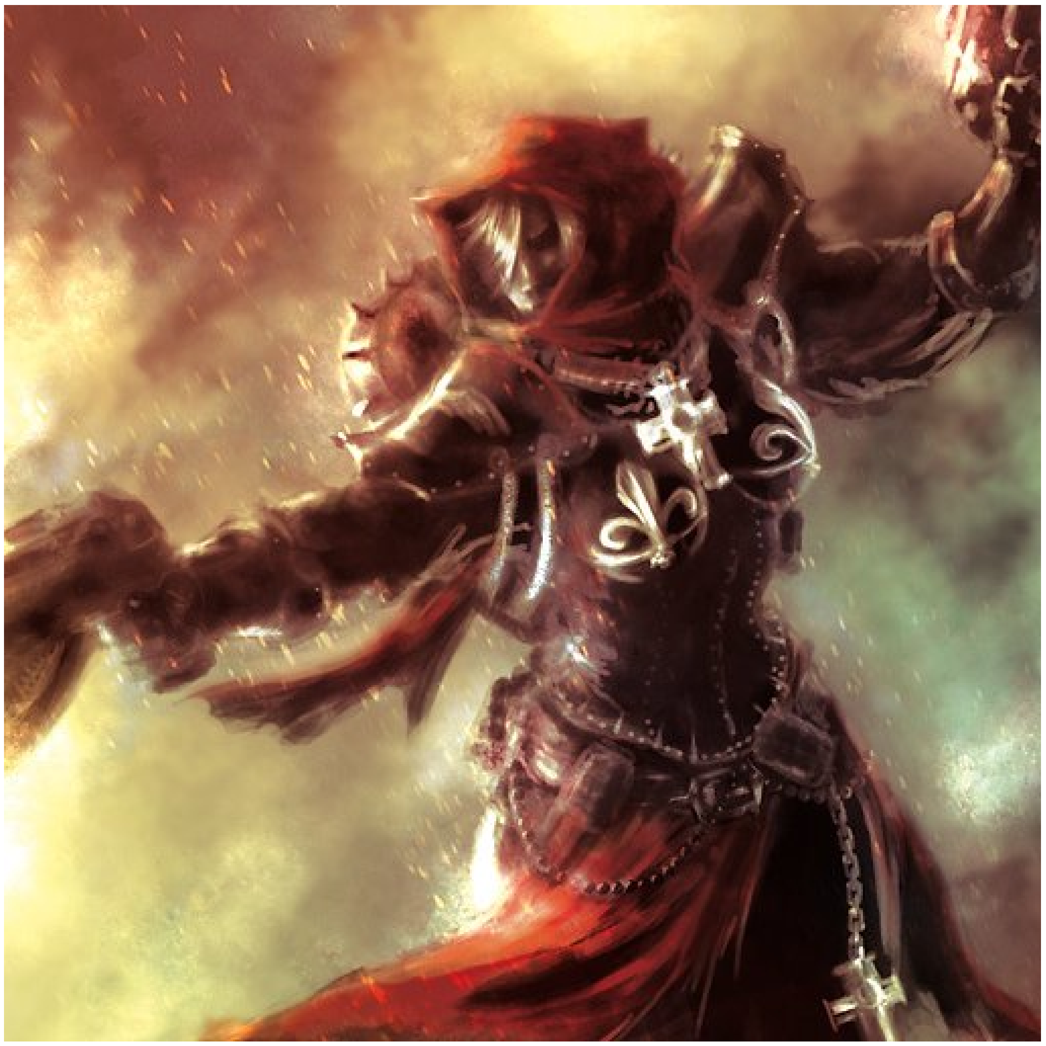


Figure 11: An artificial image which is not the best test case for Dopplegänger

The fifth image, in figure 11, is an example of hypermodern art created by artificial means yet posing the same problems as real world images. With a whole wide world of animated and artificial art out there, this is just an example to see whether our cache system can cope up with this change

6 Results

In this section, we will briefly describe the results for each image which really confirm some of the claims made by us.

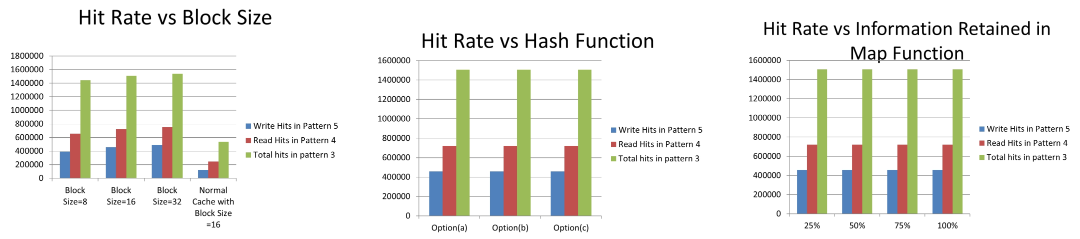


Figure 12: Plots for the Hit Rates of the the traces for (a)each block size, (b) Hash functions and (c) map functions for image 1

From the above figure 12, we can see that the hit rates are much better for the approximation cache versus the normal cache. One of the most important things to note here is that since it is a black and while image we are able to get exactly the same hit rates for both hash functions and map functions.

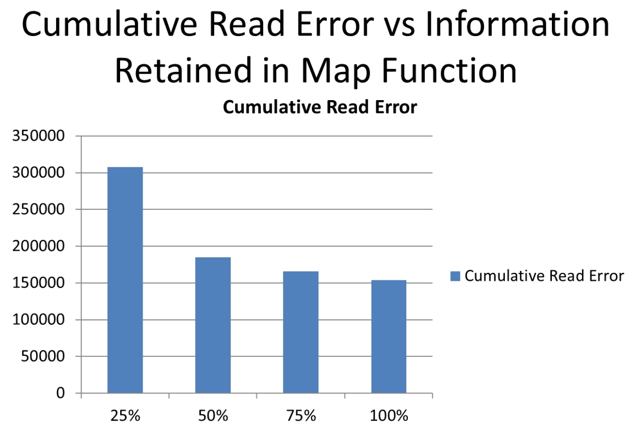


Figure 13: Plot for the cumulative read error of the the traces for each of the map functions for image 2

As you can see, the image 2 acts like one of the worst case images with the worst cumulative read error out of all 5. Also as evident in figure 13, the cumulative read error has an exponential reduction with increasing the map phase information retention. Thus we have verified our estimation that there exists a point between 0% and 100% where the tradeoff between accuracy and bits retained may become equal.

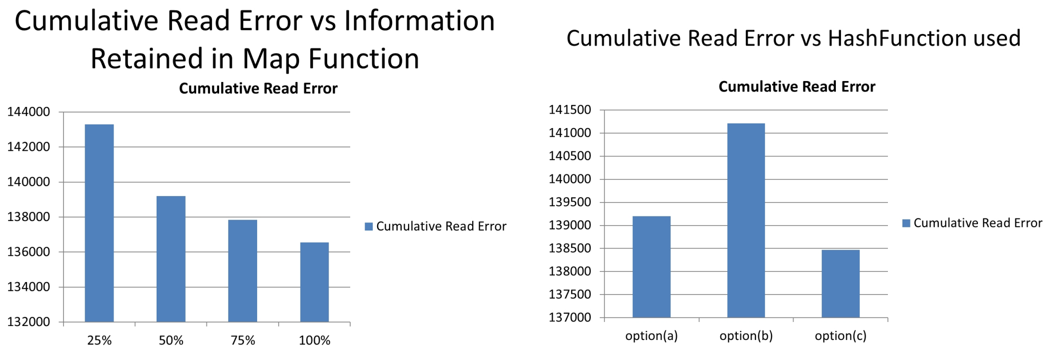
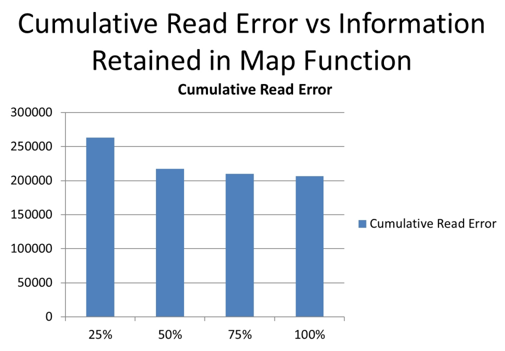


Figure 14: Plots for the cumulative read error of the the traces for (a) Hash functions and (b) map functions for image 3

Though it looks like the loss of information is steeper for a much more suited for our approximation cache, the graph is actually deceiving because of the different y-axis least count. Thus for a better image the information loss per extra bit is much lesser. Also as expected from a dark image the maximum hash function has a worse cumulative error rate and the minimum hash functions acts the best *ceteris paribus* as visible in figure 14

As is expected from image 4, the loss of information is much greater than image 3 though less than image 2 due to the presence of some similar pixel groups in the image. The loss of information is steeper than image 4 and less steeper than image 2 as is visible in figure 15. In the same figure you can also see the cumulative error is slightly lower for the maximum hash function but is excessively large for minimum hash function. Thus, it is extremely important to choose the best hashing function to reduce the loss of information with the same area and power usage.

The results for image 5 are extremely similar to that of image 2 and hence will not be discussed here. It just proves the point that not all artificially created images are suited for approximation caches and thus a more deeper analysis on the similarities in the image are required to expect the performance of a cache.



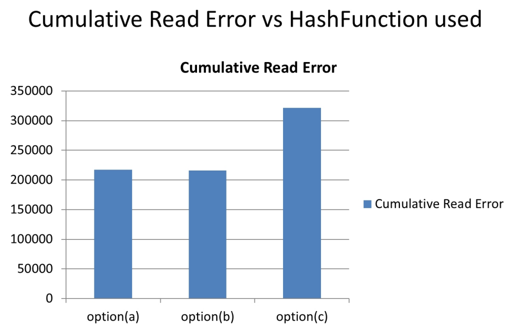


Figure 15: Plots for the cumulative read error of the the traces for (a) Hash functions and (b) map functions for image 4

7 Conclusion

In conclusion,we have seen that simulating the above cache with the developed tools is a really accurate way of testing approximation caches and check their behaviour in comparison to conventional caches. We can see that the approximation caches, though unconventional, do have a strong potential in image processing and video processing if done with proper selection of parameters to reduce the data loss and maximize the savings in terms of power and area.

This area holds a lot of potential challenges for the interested and is a ripe area of research in LLCs. Some of the potential continuations of this work could be to actually check the final status of the image, adding the functionality of uniDopplegänger to it and then customizing it to be used only for the similar part of the image.

References

[1] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger, Doppelgänger: a cache for approximate computing, in *MICRO-48* Proceedings of the 48th International Symposium on Microarchitecture