CUDA Based Speed Optimization of the PCA Algorithm

Aluno: João Gabriel Cavalcante França - 202201695

Características técnicas -

CUDA Based Speed Optimization of the PCA Algorithm

- É um paper de 2016, escrito por três autores, Salih Görgünoğlu, Kadriye Öz , Abdullah Çavuşoğlu da Universidade de Karabük, na Turquia. O paper foi publicado na revista internacional, de ciências interdisciplinares, TEM Journal.

- O paper tem como objetivo explicitar os aprimoramentos, via paralelização por CUDA, de etapas do clássico algoritmo de Análise dos Componentes Principais (PCA) , no contexto do reconhecimento facial com Eigenfaces.

- Keywords – Principal Component Analysis, CUDA, Parallel Programming, Parallel GPU Computing.

Problema

Reconhecimento Facial

- A face é um dos nossos principais focos em interações sociais. A habilidade humana no reconhecimento de faces auxiliou a nossa sobrevivência por milhares de anos, por exemplo no reconhecimento de membros de um mesmo grupo, no reconhecimento de desconhecidos e na identificação de emoções do outro.

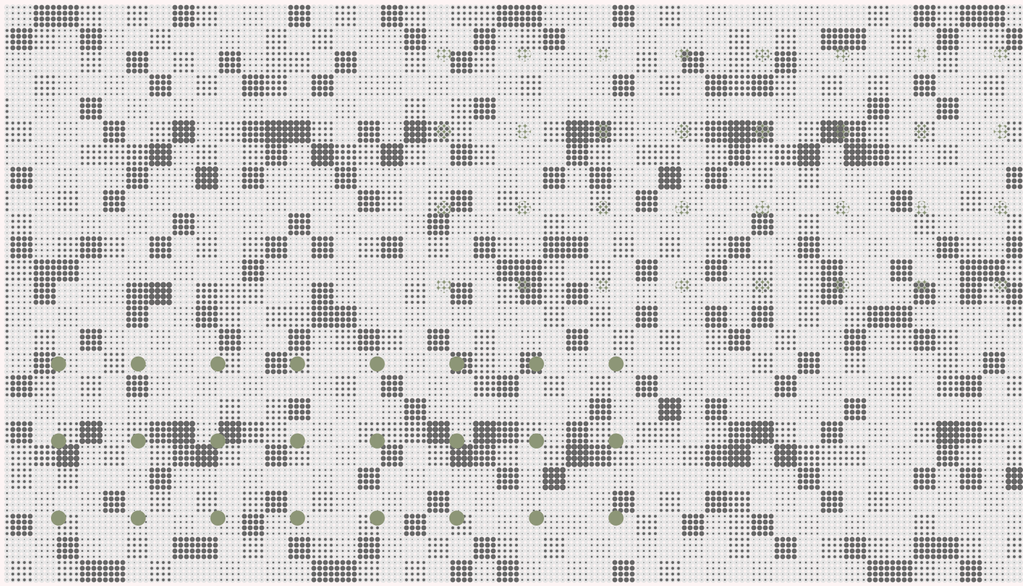
- Mas do ponto de vista computacional, o que precisamos para reconhecer uma face?
  Uma resposta simplista seria: um bom algoritmo e uma boa face, isto é, com boa resolução.

Imagens são custosas

- Imagens de alta resolução são computacionalmente custosas no sentido de armazenamento e tempo de processamento.

- Para reconhecermos faces massivamente, independente do algoritmo, precisamos de enormes quantidades de imagens de faces, em boa resolução.

- Existe uma maneira de utilizar os dados de uma imagem de forma eficiente? Reduzindo em quantidade estes dados, sem perder muita informação?

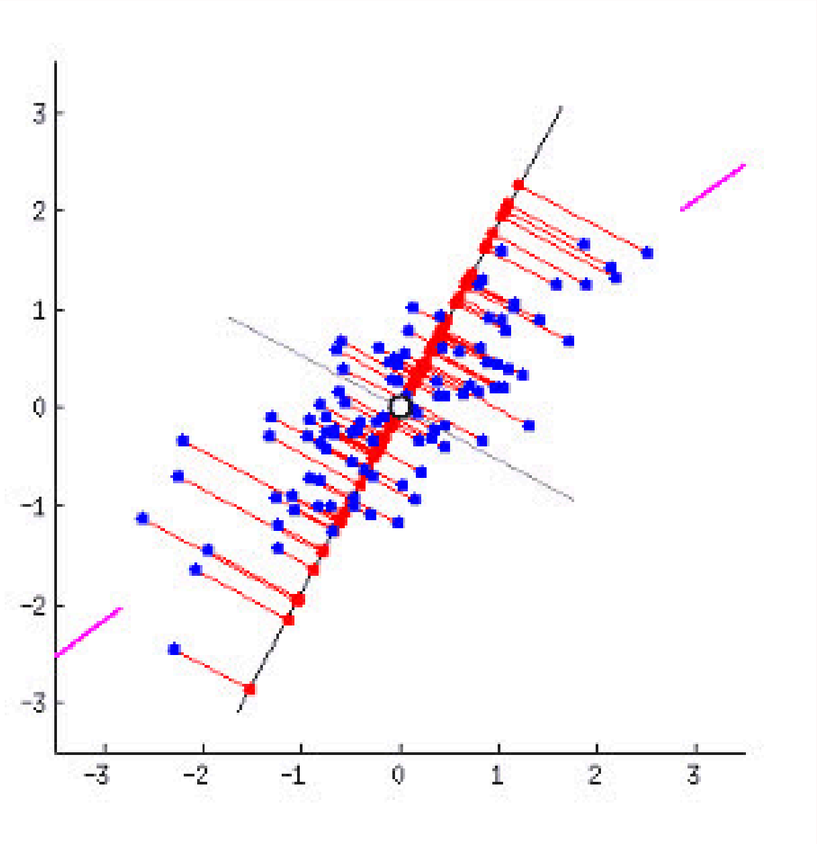


Solução Proposta

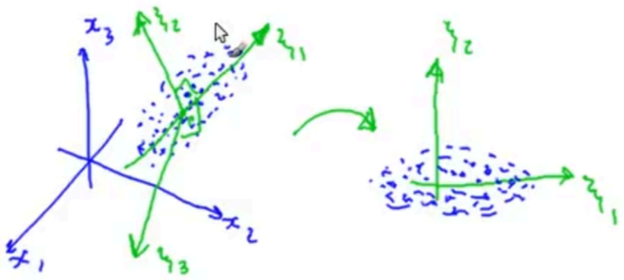
INF
INSTITUTO DE INFORMÁTICA
UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Análise de Componentes Principais (PCA)

- A análise de componentes principais é um método estatístico para redução de dimensionalidade de um conjunto de dados, preservando a máxima variabilidade, nos eixos chamados Componentes Principais.
- Os componentes principais estão em ordem de variância, isto é, o primeiro eixo a ser obtido é o de maior variância, o segundo eixo é o de segunda maior variância, e assim por diante...



Ideia matemática do PCA



PCA

$$\begin{cases} \xi_1 = r_{11}x_1 + r_{21}x_2 + \cdots + r_{n1}x_n \\ \xi_2 = r_{12}x_1 + r_{22}x_2 + \cdots + r_{n2}x_n \\ \xi_3 = r_{13}x_1 + r_{23}x_2 + \cdots + r_{n3}x_n \\ \vdots \\ \xi_n = r_{1n}x_1 + r_{2n}x_2 + \cdots + r_{nn}x_n \end{cases}$$

$$r_{11}^2 + r_{21}^2 + r_{31}^2 + \cdots r_{n1}^2 = 1$$

$$r_{12}^2 + r_{22}^2 + r_{32}^2 + \cdots + r_{n2}^2 = 1$$

$$\vdots$$

$$\vec{r}_j^T \vec{r}_j = 1$$

$$\vec{r}_j = (r_{1j}\ r_{2j} \cdots r_{nj})^T$$

$$Var(\xi_1) = var(r_{11}x_1 + r_{21}x_2 + \cdots + r_{n1}x_n)$$

$$= var(\vec{r}_j \vec{x}) = \vec{r}_j^T \Sigma \vec{r}_j$$

$$= var(\vec{r}_1 \vec{x}) = \vec{r}_1^T \Sigma \vec{r}_1$$

$$\arg_{\vec{r}_1}\left[\max(\vec{r}_1^T \Sigma \vec{r}_1)\right]$$

sujeito a: $\vec{r}_1^T \cdot \vec{r}_1 = 1$

Em matemática, em problemas de otimização, o método dos multiplicadores de Lagrange permite encontrar extremos (máximos e mínimos) de uma função de uma ou mais variáveis suscetíveis a uma ou mais restrições.[2]

Por exemplo (veja a figura 1 à direita), considere o problema de otimização

maximize $f(x, y)$, ou seja, deseja-se encontrar o ponto máximo desta função sujeito a $g(x, y) = c$.

O método consiste em introduzir uma variável nova (λ, normalmente), chamada de multiplicador de Lagrange. A partir disso, estuda-se a função de Lagrange, assim definida:

$$\Lambda(x, y, \lambda) = f(x, y) - \lambda \cdot \big(g(x, y) - c\big),$$

$$\vec{\nabla}\left(\vec{r}_1^T \Sigma \vec{r}_1 + \lambda(1 - \vec{r}_1^T \vec{r}_1)\right) = 0$$

$$\vec{\nabla}(\vec{r}_1^T \Sigma \vec{r}_1) - \vec{\nabla}(\lambda \vec{r}_1^T \vec{r}_1) = 0$$

$$\Sigma \vec{r}_j + \Sigma^T \vec{r}_j - \vec{\nabla}(\lambda \cdot \vec{r}_1 \mathbb{1} \vec{r}_1)$$

$$2\Sigma \vec{r}_j - 2\lambda \vec{r}_j = 0$$

Conclusão: precisamos apenas calcular os autovalores da matriz de variância/covariância!

$$\Sigma \vec{r}_j = \lambda \vec{r}_j$$

equação de auto valores e auto-vetores

$$\arg\left[\max\left[var(\xi_1)\right]\right] =$$

$$var(\xi_1) = \vec{r}_1^T \Sigma \vec{r}_1$$

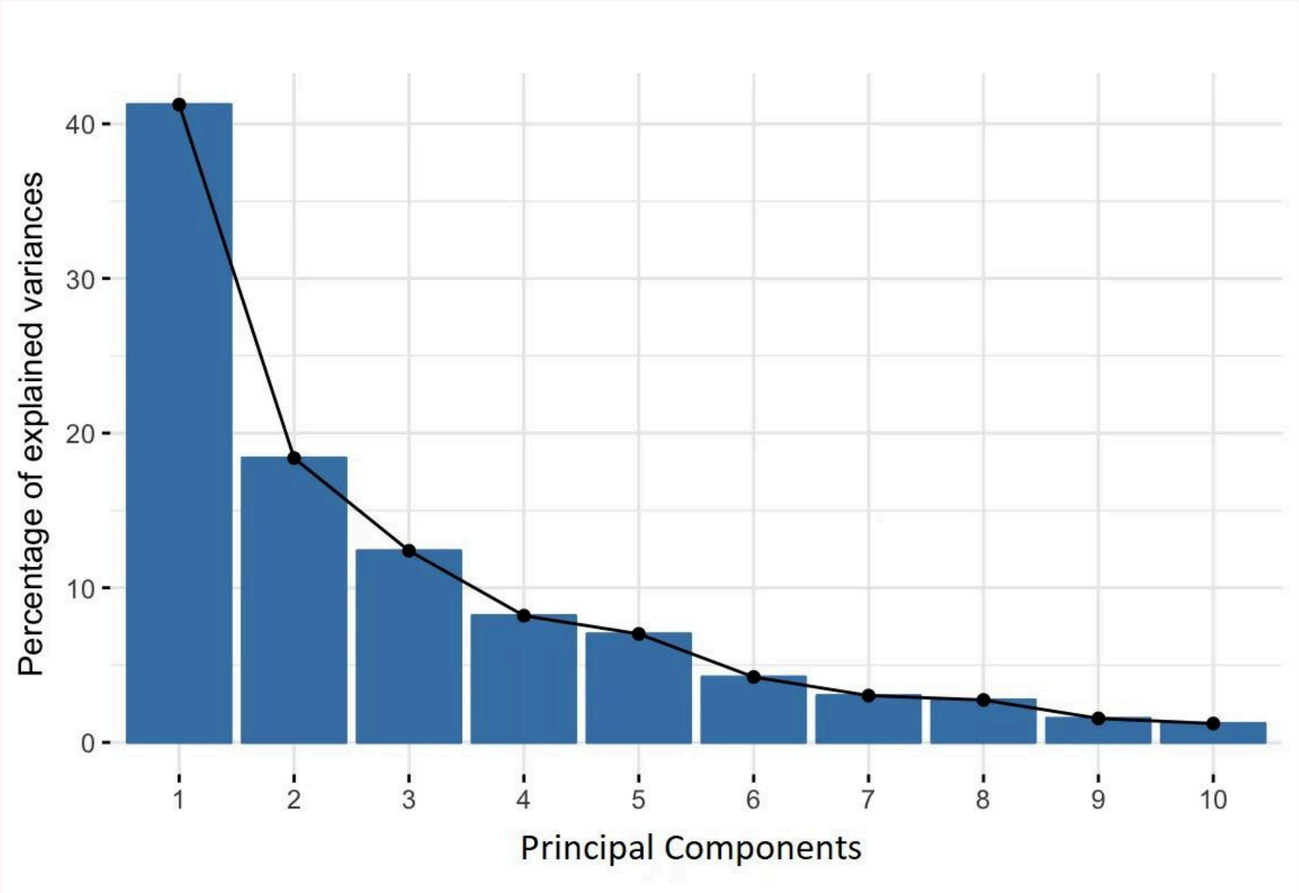
$$= \vec{r}_1^T \lambda \vec{r}_1$$

$$= \lambda \cancel{\vec{r}_1^T \vec{r}_1}^{\,1}$$

$$\max\left[var(\xi_1)\right] = \lambda$$

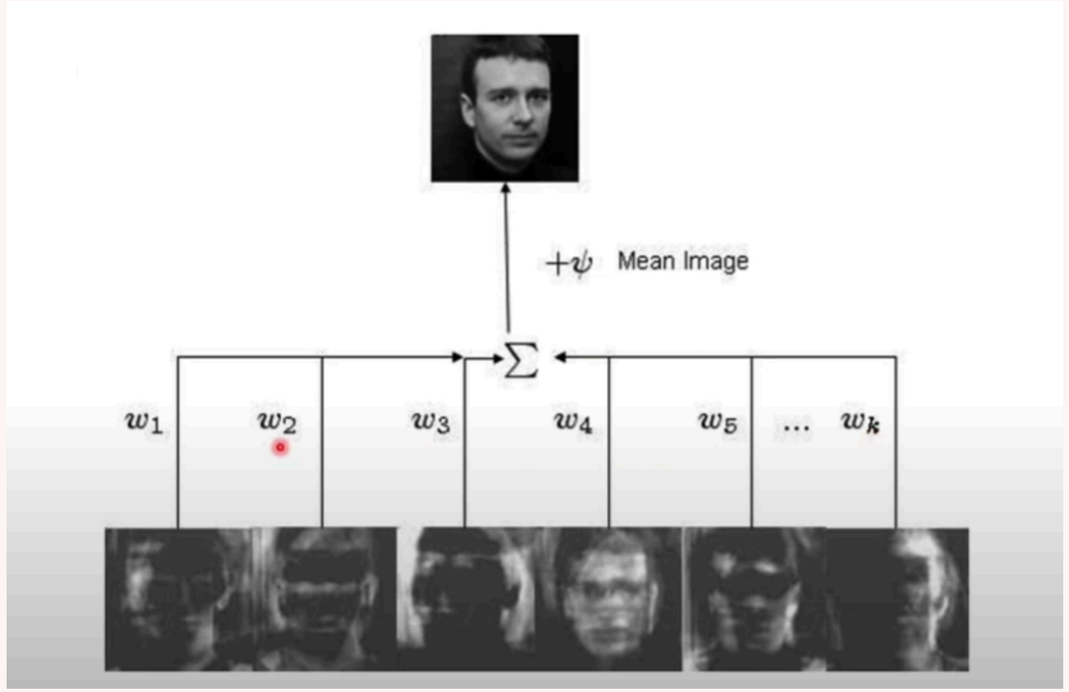
INF INSTITUTO DE INFORMÁTICA

UFG UNIVERSIDADE FEDERAL DE GOIÁS


Percentage of explained variances
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10
Principal Components

Relação Eigenfaces x PCA

- O método Eigenfaces, proposto por Matthew Turk e Alex Pentland, em 1990 seleciona as imagens de maior variância do treinamento, a partir de uma matriz com todas as imagens. A projeção destas imagens de maior variância nas componentes principais são chamadas de Eigenfaces. Cada face do dataset é uma soma ponderada das Eigenfaces.



- É dividido em duas etapas principais, que são subdivididas em etapas menores: Extração do Vetor de Características do Dataset/Reconhecimento

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Etapas do Eigenfaces

Table 1.Calculation Steps Of Feature Extraction Using The PCA Algoritm

Algorithm	Description	Explanation
Normalization	$x' = \frac{x - \min_A}{\max_A - \min_A}$	x represents the color value of a pixel in the image, $\min_A$ represents the smallest color value in the face (i.e. image), $\max_A$ represents the biggest color value in the face (i.e. image),
Mean Calculation	$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$	Γ_1, Γ_2.. Γ_M column matris corresponding to each face in the training set, if the size of the face image is NxN then each column matris Γ will have the size N^2x1, M number of faces in the training set, Ψ mean value of the set
Φ calculation	$\Phi_i = \Gamma_i - \Psi$	Φ_i the difference of the face from the mean value,
A Transpose Calculation	A^T	A=[$\Phi_1,\Phi_2,\Phi_3...,\Phi_M$], Φ_i fort he size N^2x1, The A matris size becomes N^2xM,
L Calculation	$L = A^T A$	L reduced matris (sizewise) calculated instead of covariance matris, multiplication of MxN^2 with N^2xM sized matris produces MxM sized matris,
Eigenvector Calculation	$L.v = \lambda v$	v is eigenvector and λ is eigenvalue
Eigenface Calculation	$u_l = \sum_{k=1}^{M} v_{lk}\Phi_k$ $l = 1, \ldots, M$	U is eigenface,
Feature Vector Calculation	$w_k = u_k^T(\Gamma - \Psi)$	W is the feature vector

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T$$

Table 2. Calculation Steps Of Face Recognition Using The PCA Algoritm

Algorithm	Description	Explanation
Normalization	$x' = \frac{x - \min_A}{\max_A - \min_A}$	the color values of the face to be recognized normalized to between 0-1,
Difference from mean	$\Phi_T = \Gamma_T - \Psi$	its difference from the mean face is calculated
Feature vector extraction	$w_t = U.\Phi_T$	The feature vector is calculated using the eigenvectors,
Distance control	$d_i = \sum_{j=1}^{M}\lvert w_T(j) - w_i(j)\rvert$ $i = 1,2,\ldots N$	The distance between the feature vectors are calculated for a successful matching,

- Evita com que trabalhemos com uma matriz NxN, ao invés de usar ela, utilizamos uma MxM, já que M < N² e o posto da matriz N²xM é no máximo M.

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Normalização

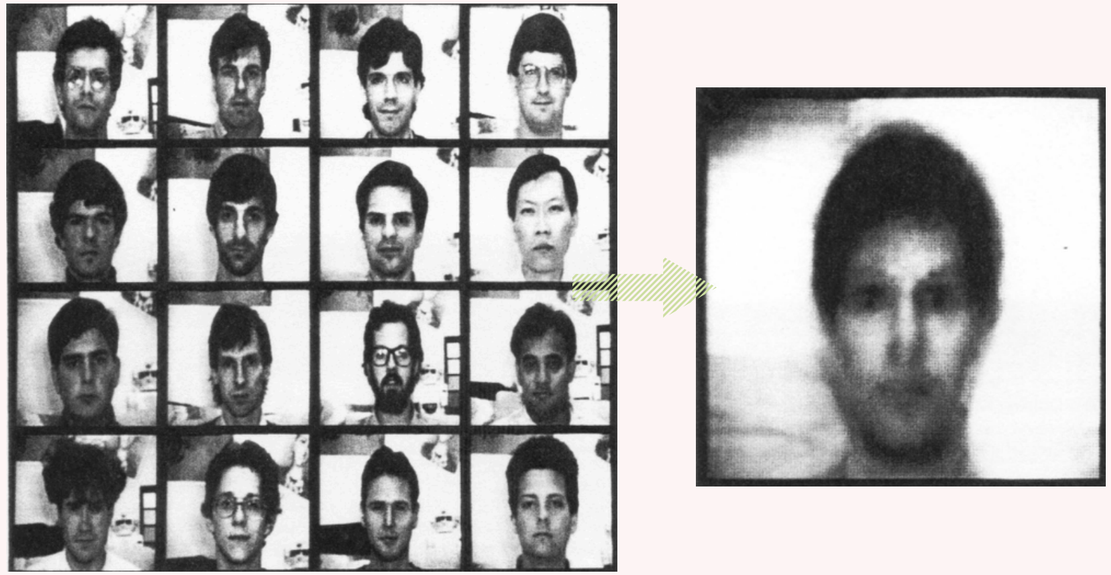
- Na etapa de Normalização min-max, os autores descreveram que cada thread percorreu os pixels que formam a imagem, encontrando os valores mínimos e máximos. E a mesma thread realiza uma segunda operação que, de fato, normaliza os valores da imagem dentro do intervalo de 0 a 1.

$$x' = \frac{x - \min_A}{\max_A - \min_A}$$

Cálculo da Face Média

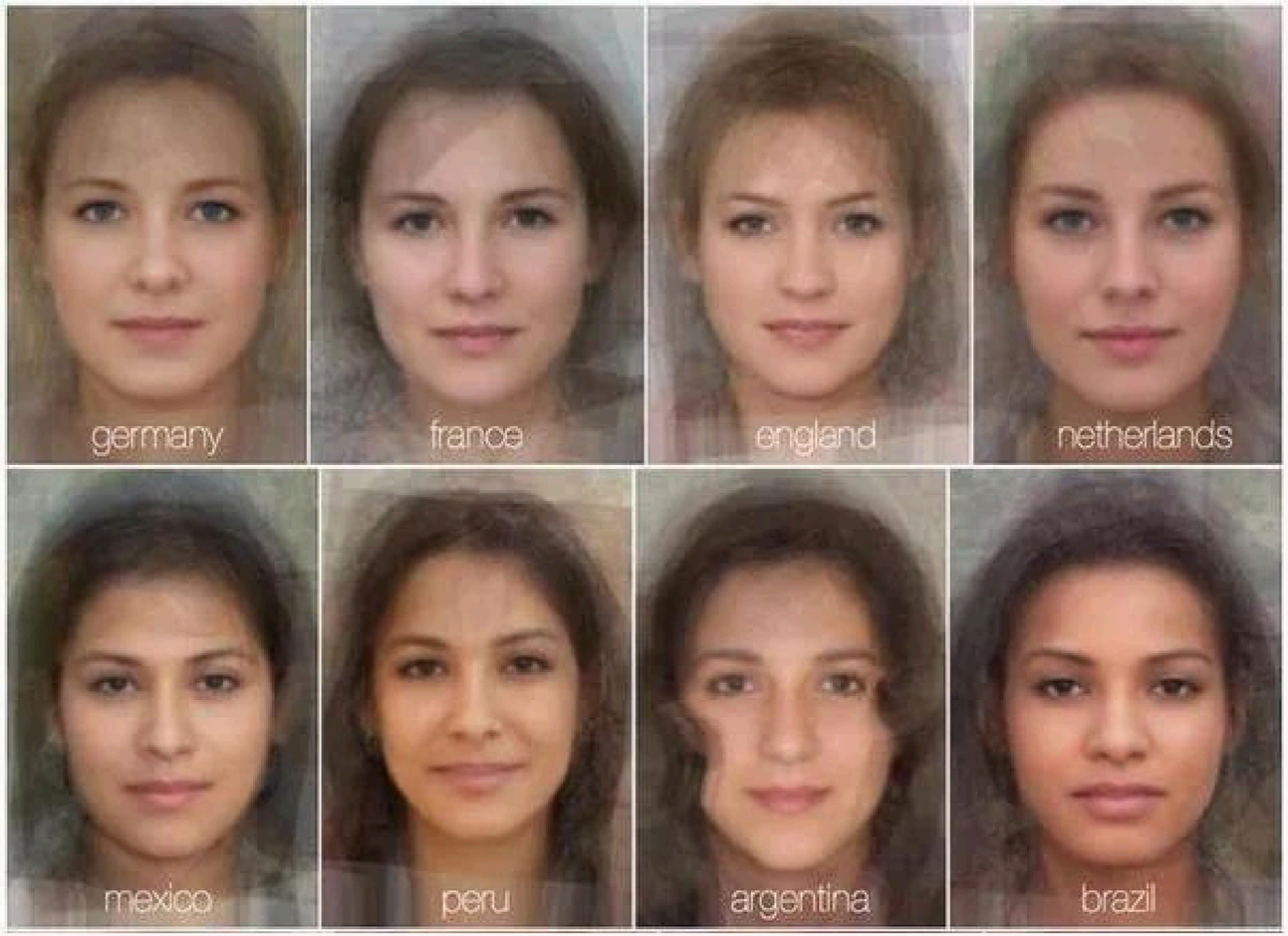
- O cálculo da face média é a etapa em que transformamos cada imagem NxN do nosso dataset (de M imagens) em uma matriz-coluna de tamanho N^2, calculamos a média destas M matrizes-coluna e colocamos em uma matriz Ψ, que será utilizada no cálculo da diferença entre as faces ($\Phi i = \Gamma i - \Psi$)

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$$



INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Cálculo da Face Média



Cálculo do vetor de Diferenças

$$\Phi_i = \Gamma_i - \Psi$$

- É a diferença entre a face de índice i e a média

$$\Phi i = \Gamma i - \Psi$$

- Tem dimensões N^2x1 e formará a matriz A, alvo principal da Análise dos Componentes Principais, em que reduziremos sua dimensionalidade de N^2xM para M eigenvectors (analisando a matriz reduzida), que transformaremos depois em eigenfaces

$$A=[\ \Phi 1,\Phi 2,\Phi 3...,\Phi M],$$

N^2xM

Algorithm 1: The definition of the device and funtion to be executed by the device.

```
__global__ void calculateA(float* d_A,float* d_face,
float* d_meanface, int facecount,int pixel)
{
    int i=threadIdx.x;
    int j = blockIdx.x;
    d_A[index] = d_face[index] - d_meanface[j];
}   int index=i+j*facecount;
```

Algorithm 2: Memory allocation on the device and copying the data to this memory space.

```
unsigned int size_face = faceCount * pixel;
unsigned int mem_size_face = sizeof(float) *
size_face;
unsigned int mem_size_meanface = sizeof(float)
*pixel;
float* d_A;
float* d_face;
float* d_meanface;
cutilSafeCall(cudaMalloc((void**) &d_A,
mem_size_face));
cutilSafeCall(cudaMalloc((void**) &d_face,
mem_size_face));
cutilSafeCall(cudaMalloc((void**) &d_meanface,
mem_size_meanface));
cutilSafeCall(cudaMemcpy(d_face,face,
mem_size_face, cudaMemcpyHostToDevice) );
cutilSafeCall(cudaMemcpy(d_meanface,meanface,
mem_size_meanface, cudaMemcpyHostToDevice) );
```

Algorithm 3: Execution of the codes on the device, transferring the results back to the host and freeing the spaces on the device.

```
int block =pixel;
int thread=faceCount;
calculateA <<< block, thread
>>>(d_A,d_face,d_meanface,faceCount,pixel);
cudaThreadSynchronize();
cutilSafeCall(cudaMemcpy(A, d_A,
mem_size_face,cudaMemcpyDeviceToHost) );
cutilSafeCall(cudaFree(d_A));
cutilSafeCall(cudaFree(d_face));
cutilSafeCall(cudaFree(d_meanface));
```

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Cálculo da Matriz Reduzida

Algorithm 4: Calculating L=ATxA on CPU

```
// faceCount:the number of face image ;
// pixel: the pixel number of a face image;
 for i←0 to faceCount
 for j←0 to faceCount
 for k←0 to pixel
            L[i, j] ←. L[i, j] +ATranpose[i, k] *
A[k, j];
 end of for;
 end of for;
```

- A partir da construção da matriz A, composta por M vetores Φ, de dimensões N²xM criamos a matriz reduzida L = A^T * A

Algorithm 5: Calculating $L=A^TxA$ based on the register of the GPU

```
// threadDim: the dimension of the thread in each block;
// blockDim: the dimension of the block in each grid;
// blockIdx.x: the current block ID;
// threadIdx.x: the current thread ID;
// faceCount:the number of face image ;
// pixel: the pixel number of a face image;
//index: the adress of data which willl be calculated
//aindex: index which will be used in multiplication
//aTIndex: index of the cell in transpose matrix( AT) which
will be used in multiplication
1.  index=blockIdx.x*faceCount+threadIdx.x;
2.  for j=0 to pixel
3.          aIndex←blockIdx.x*pixel+j;
4.          aTIndex←j*faceCount+threadIdx.x;
5.          sum←sum+ A[aIndex]*d_AT [aTIndex];
6.  end of for;
7.  L[index]← sum;
```

Cálculo da Eigenface

- Os eigenfaces são combinações lineares entre os autovetores da matriz de covariância, ou da matriz reduzida e os vetores de diferença Φ

Following this analysis, we construct the M by M matrix $L = A^T A$, where $L_{mn} = \mathbf{\Phi}_m^T \mathbf{\Phi}_n$, and find the M eigenvectors, $\mathbf{v}_l$, of L. These vectors determine linear combinations of the M training set face images to form the eigenfaces $\mathbf{u}_l$.

$$\mathbf{u}_l = \sum_{k=1}^{M} \mathbf{v}_{lk} \mathbf{\Phi}_k, \qquad l = 1, \ldots, M \qquad (6)$$

Cálculo do Vetor de Características

- Vetor, de dimensões reduzidas, para a etapa de reconhecimento, no cálculo da distância entre os rostos (distância entre features).

$$w_k = u_k^T (\Gamma - \Psi)$$

Principais resultados

INF
INSTITUTO DE INFORMÁTICA
UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Especificações Técnicas e Considerações

- **Configuração Base**

Intel Core I3 550 3.2 GHz CPU, 4 GB DDR3 1333 MHz RAM, Microsoft Windows 7 Professional (x64) O/S

- **Procedimento**

Os testes foram realizados, em cada etapa, em dez execuções, tanto na GPU, quanto na CPU. Depois, somados estes tempos e feito o cálculo da média individual de cada uma das etapas de Treinamento (Tabela 1.) e Reconhecimento (Tabela 2.)

- **Dataset**

Utilizaram o ORL face database para aplicação do PCA no Eigenfaces, em 400 imagens grayscale de 92x112 com 40 indivíduos diferentes.

Table 3.The specification of graphics cards used in PCA computations.

	GeForce GT220	GeForce GT630	GeForce 450
Core Number	48	96	192
Compute Capability	1.2	2.1	2.1
Streaming Multiprocessors	6	2	4
Max threads per blocks	512	1024	1024
Clock rate	1012000	1620000	118900

$$speedup = \frac{\frac{1}{10}\sum_{n=1}^{10} TempoCPU}{\frac{1}{10}\sum_{n=1}^{10} TempoGPU}$$

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Visão Geral dos Testes

- **Testes**

O paper apresentou testes apenas para a extração do vetor de características.

- **Speedups**

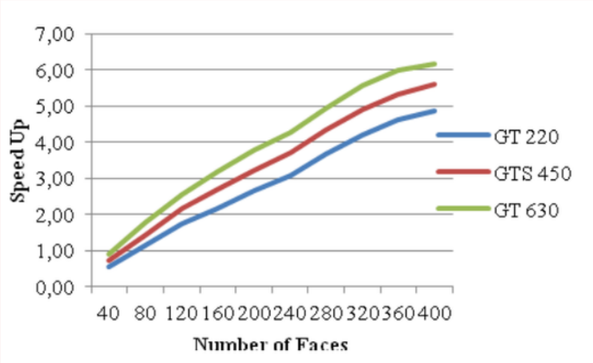
Nas quatro primeiras etapas, o speedup foi relativamente baixo (1,2 - 12x), enquanto nas três últimas etapas o speedup foi alto (1,5 - 250x)

- 1. Normalização
- 2. Face Média
- 3. Diferença Φ
- 4. Transposição dos vetores Φ
- 5. Cálculo da matriz reduzida
- 6. Cálculo da Eigenface
- 7. Cálculo do Vetor de Características

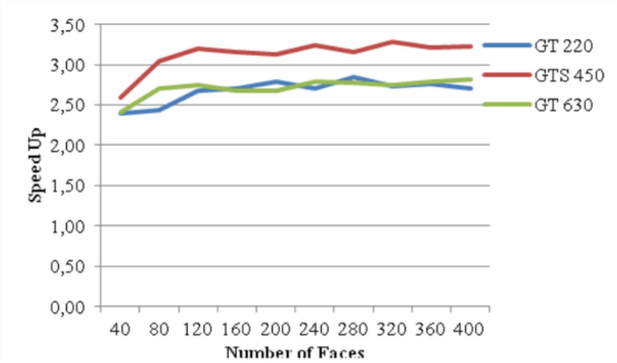
INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Etapas de menor speedup

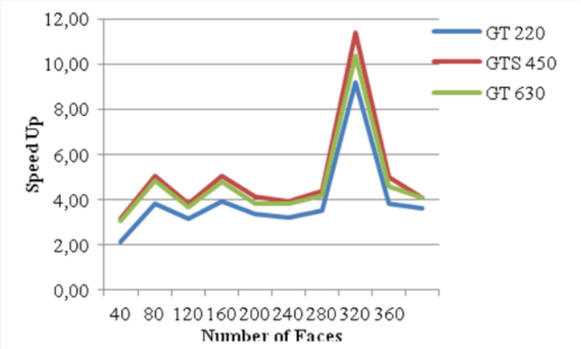
1. Normalização



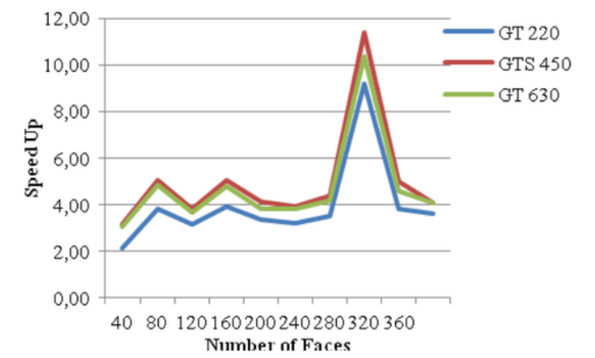
2. Cálculo da Face Média



3. Cálculo da Diferença Φ

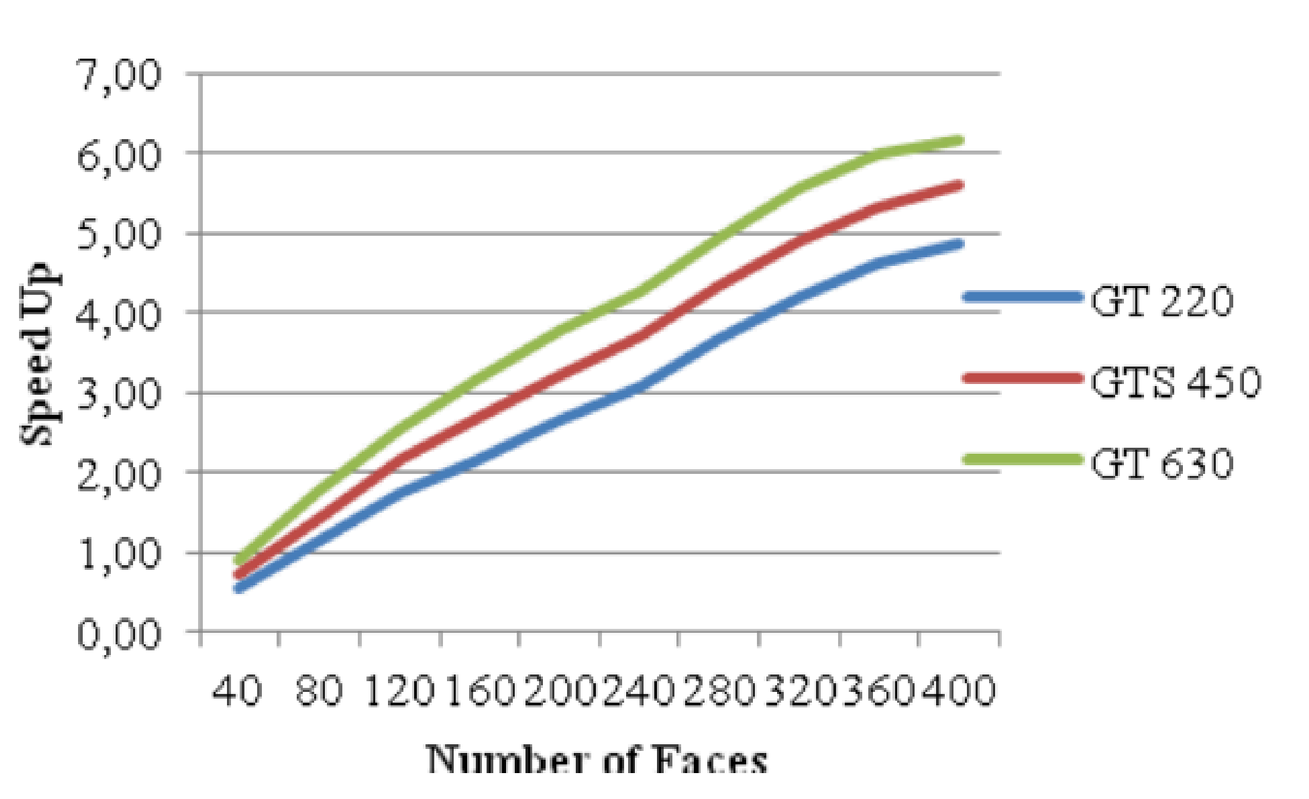


4. Transposição da matriz de diferenças



INF INSTITUTO DE INFORMÁTICA
UFG UNIVERSIDADE FEDERAL DE GOIÁS

Normalização



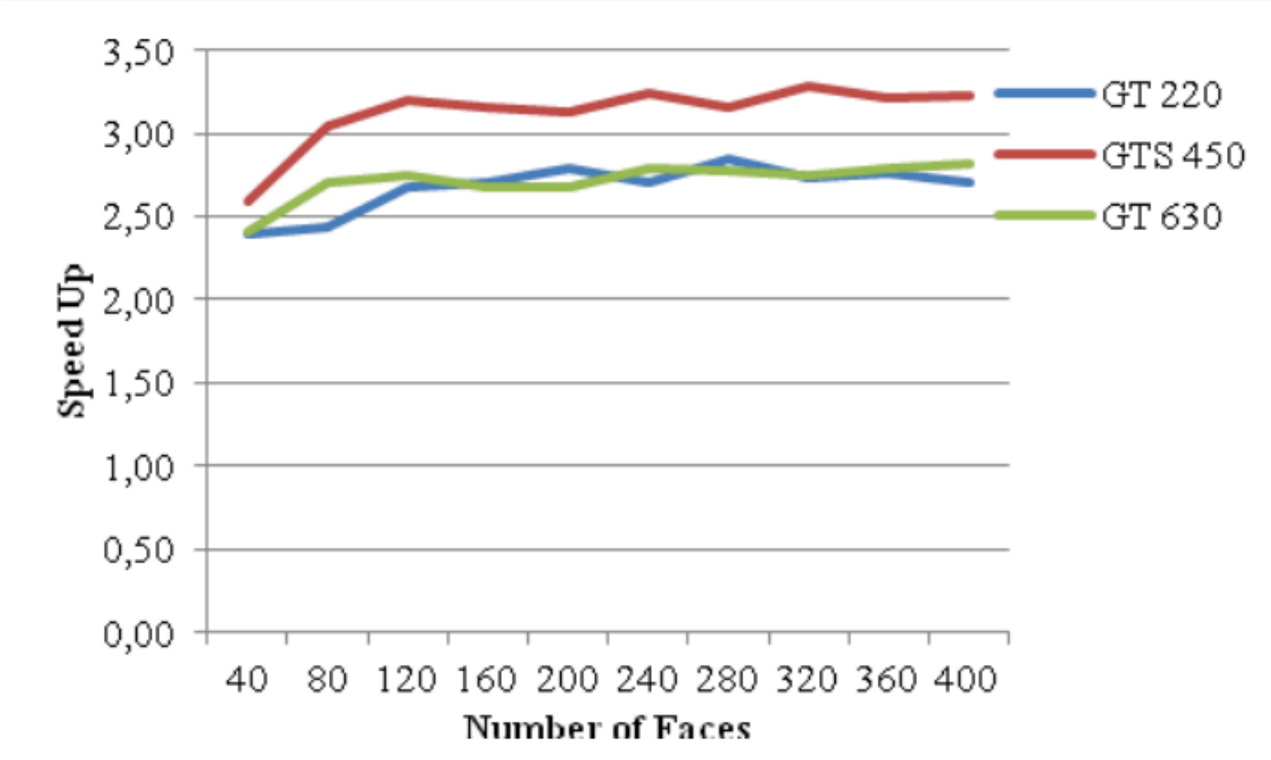
$$x' = \frac{x - \min_A}{\max_A - \min_A}$$

- Speedups de ~1 até ~6
- GT 630 com resultados superiores
- Resultados melhores com quantidades maiores de faces

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Cálculo da Face Média Paralelizada

- Lembrando que o cálculo da face média é a etapa em que transformamos cada imagem NxN do nosso dataset (de M imagens) em uma matriz-coluna de tamanho N^2, calculamos a média destas M matrizes-coluna e colocamos em uma matriz Ψ, que será utilizada no cálculo da diferença entre as faces ($\Phi i = \Gamma i - \Psi$)
- Os autores, apesar de não apresentarem o código, citam que durante este processo, na GPU, foi contabilizado o tempo de passagem de dados das imagens da CPU para a GPU.



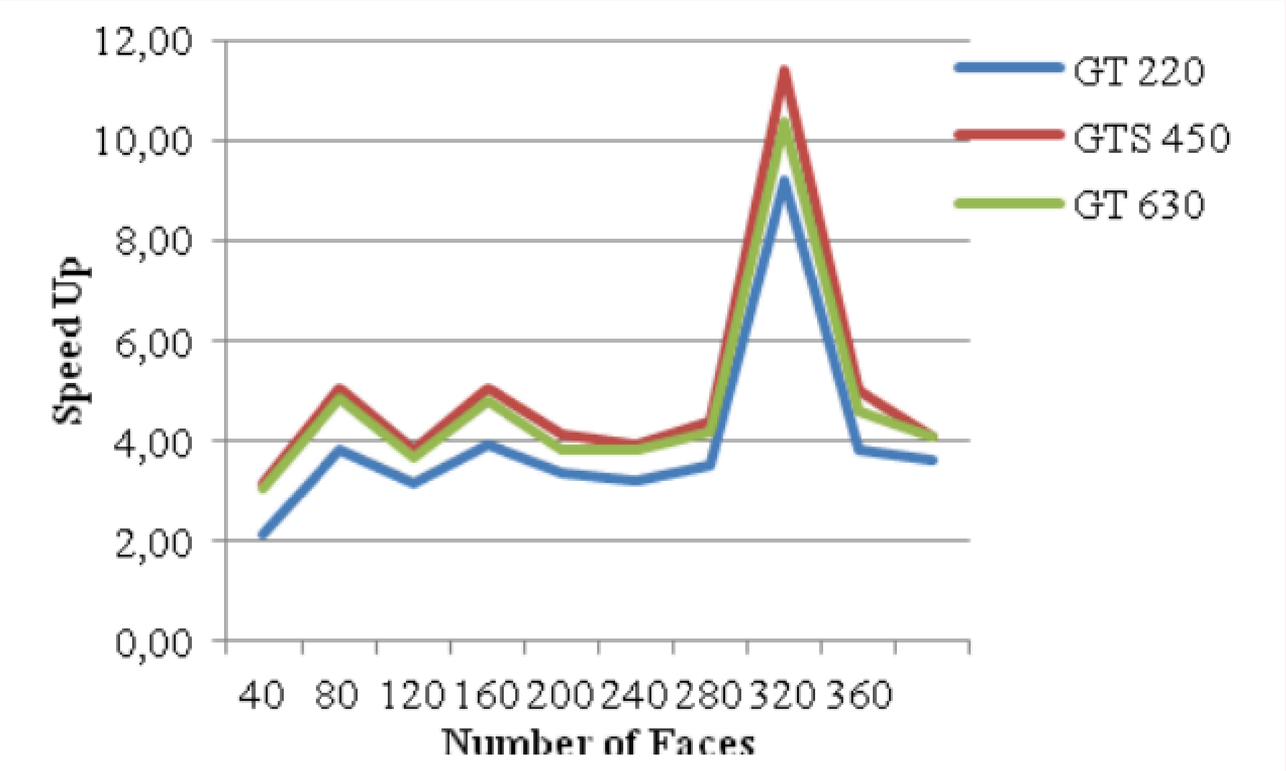
- Speedups de ~2,5 até ~3,25
  - GT 450 com resultados superiores
    - A quantidade de faces não influencia muito na performance nesta etapa

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$$

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

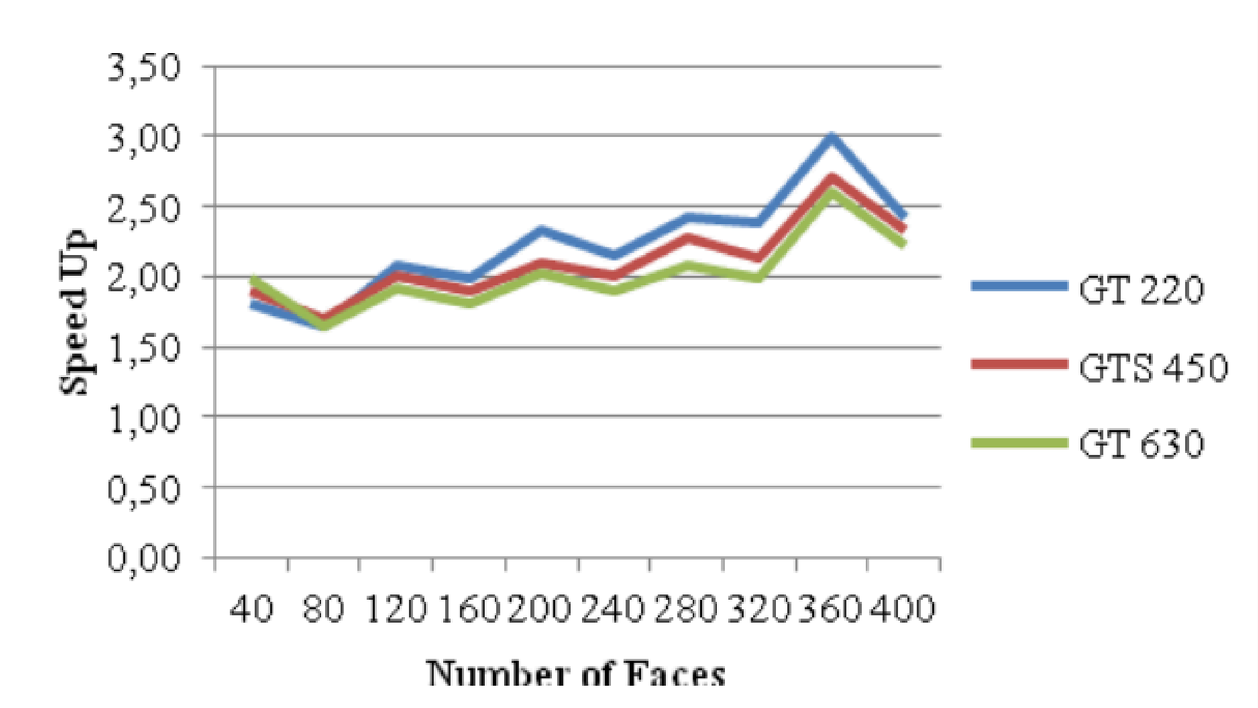
Cálculo da Diferença Φ

- O autor cita que por volta de 320 imagens, existe uma conformidade exata entre o número de núcleos e o número de threads quando as divisões dos dados das imagens são feitas nas threads, assim obtendo um speedup mais alto.
- Speedups de ~2 a ~12
- GT 450 com resultados superiores
- A quantidade de faces também não influencia muito nesta etapa (para quantidades pequenas)



$$\Phi_i = \Gamma_i - \Psi$$

Transposição da matriz de diferenças

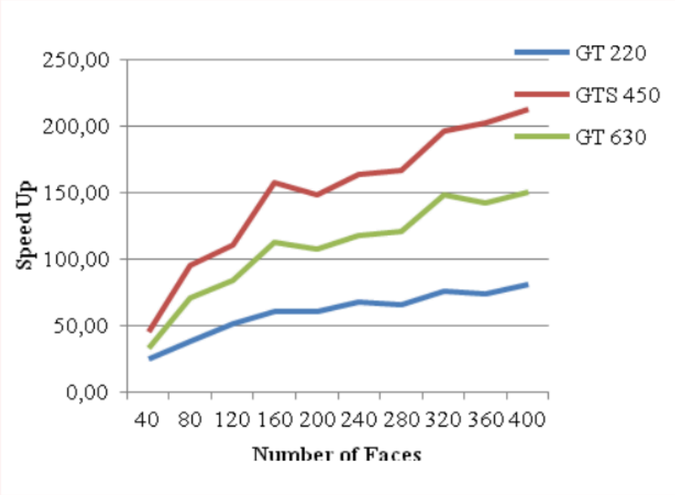


- Speedups de ~1,75 até ~3
- GT 220 com resultados superiores

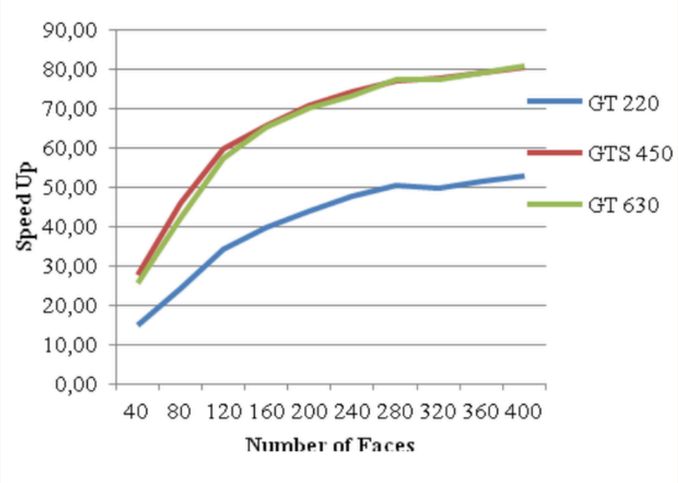
INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Etapas de maior speedup

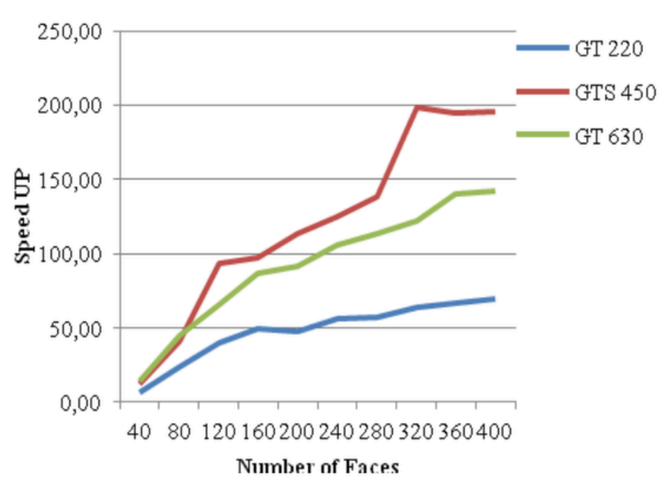
1. Cálculo da Matriz Reduzida (L)



2. Cálculo dos Eigenfaces (U)

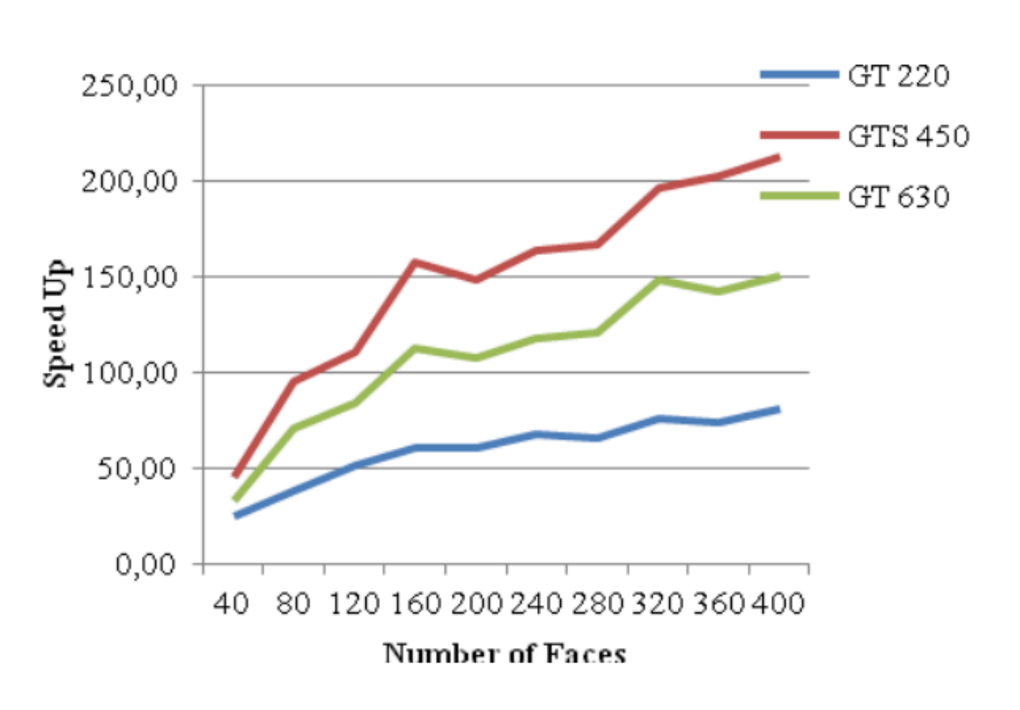


3. Cálculo do Vetor de Características (W)



Cálculo da Matriz Reduzida (L)

- É uma das etapas mais custosas em tempo de todo o algoritmo do Eigenfaces.
- O aumento da quantidade de núcleos nas placas gera um efeito positivo de performance
- Superioridade da GT 450



$$L = A^T A$$

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Cálculo das Eigenfaces (U)

- Speedsups de ~15 a ~80x
- A diferença entre as placas com mais núcleo para a com menos núcleos é expressiva.
- A quantidade de faces também é um fator importante, quanto mais faces, maior o speedup observado.

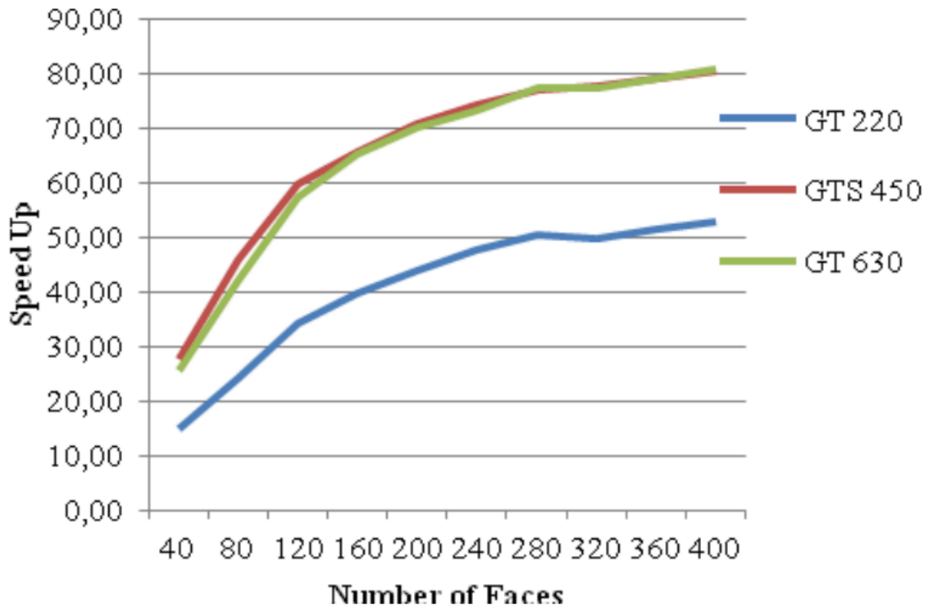
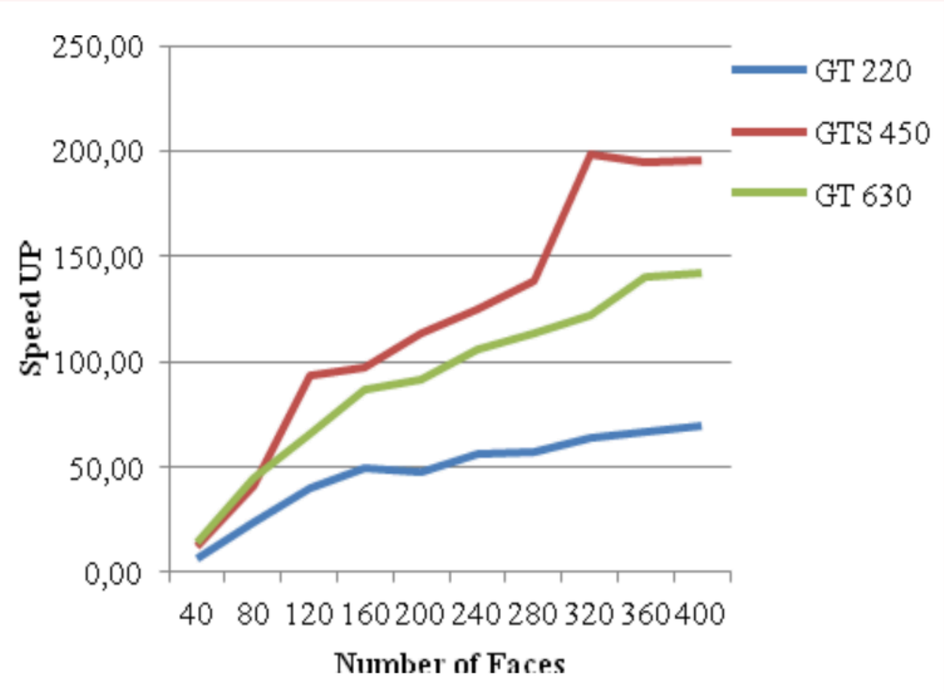


Figure 8.Speedup rate of Eigenface (U) calculation

$$u_l = \sum_{k=1}^{M} v_{lk} \Phi_k$$
$$l = 1, \ldots, M$$

Cálculo do Vetor de Características (W)

- Speedsups de ~5 a ~185x
- O aumento da quantidade de núcleos nas placas gera um efeito positivo de performance
- O aumento da quantidade de núcleos nas placas gera um efeito positivo de performance



- Apesar da GT220 ter apenas 48 cores, metade das outras placas, o desempenho do GT220 é competitivo e eficiente, graças ao maior número de processadores de streaming (SMs), que permitem processar múltiplos blocos simultaneamente. Essa vantagem se destaca especialmente quando os blocos de dados são pequenos, pois os SMs conseguem lidar com mais tarefas simultâneas, mantendo o desempenho competitivo com GPUs que possuem mais núcleos.

Table 3.The specification of graphics cards used in PCA computations.

	GeForce GT220	GeForce GT630	GeForce 450
Core Number	48	96	192
Compute Capability	1.2	2.1	2.1
Streaming Multiprocessors	6	2	4
Max threads per blocks	512	1024	1024
Clock rate	1012000	1620000	118900

Table 4. Time analysis for 400 images .

Algorithms	**CPU Time (ms)**	**GPU Time (GT 220) (ms)**	**GPU Time (GT 630) (ms)**	**GPU Time (GTS 450) (ms)**	**Speed Up***
Normalization	135.27	29.60	21.89	24.26	5.57
Mean calculation	35.57	13.17	12.65	11.02	3.22
Φ(A) calculation	83.90	23.07	20.51	20.64	4.06
A Transpose calculation	67.10	27.62	30.14	28.84	2.32
L calculation	35576.11	442.28	237.00	167.30	212.64
U calculation	5584.81	105.86	69.23	69.49	80.36
W calculation	7099.00	101.45	50.05	36.26	195.78
Overall	**48446.49**	**743.05**	**428.82**	**346.79**	**139.69**

*The values correspond to this configuration with the GTS 450 graphics card.

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

Conclusões

- O artigo demonstrou bons resultados, com speedups expressivos de 2-250x, através da paralelização da Análise dos Componentes Principais (PCA).
- O que demonstra que o ganho performático da paralelização do PCA é válido, principalmente nas etapas mais custosas, como no cálculo da matriz reduzida, no cálculo dos Eigenfaces e no cálculo do Vetor de Características. Nas 4 primeiro etapas, o ganho é limitado.
- O artigo omite trechos de código ou pseudo-códigos triviais para a melhor compreensão do leitor. Além disso, não existem testes para a etapa de Reconhecimento, o foco é exclusivo no Treinamento.

Referências

CUDA Based Speed Optimization of the PCA Algorithm - https://www.temjournal.com/content/52/TemJournalMay2016_152_159.pdf

Turk M., Pentland A. (1991). Eigenfaces for recognition. Journal of Cognitive Neuroscience. 3, 71-86 - https://www.face-rec.org/algorithms/PCA/jcn.pdf

Reconhecimento de Padrões - USP - (PCA) Análise de Componentes Principais- https://youtu.be/2MySwcqdi2A?list=PLF2A972E390252A13

Reconhecimento de padrões — Eigenfaces - https://medium.com/@williangpessoa/reconhecimento-de-padr%C3%B5es-eigenfaces-e4cef8f04919#:~:text=Eigenfaces%20%C3%A9%20o%20conjunto%20de,(sigla%20PCA%20em%20ingl%C3%AAs).