Paralelização em CPU de um Extrator de Features "Histogram of Oriented Gradients"

Gustavo de Mello Crivelli – RA 136008
João Pedro Ramos Lopes – RA 139546

Curso de Engenharia de Computação – Universidade Estadual de Campinas – Campus de Campinas
13081-970 – Campinas – SP – Brasil

g136008@dac.unicamp.br, j139546@dac.unicamp.br

Paralelização em CPU de um Extrator de Features "Histogram of Oriented Gradients"

Gustavo de Mello Crivelli, João Pedro Ramos Lopes

Curso de Engenharia de Computação – Universidade Estadual de Campinas – Campus de Campinas
13081-970 – Campinas – SP – Brasil

g136008@dac.unicamp.br, j139546@dac.unicamp.br

Abstract. *Histogram of Oriented Gradients-based feature extractors are increasingly common in modern object detection algorithms. In this report, we show the process of designing and implementing a parallel CPU-grounded HOG extractor, which can be up to XX.X times faster than its serial equivalent.*

Resumo. *Extratores de features de imagens baseados em Histogramas de Gradientes Orientados são bastante utilizados em algoritmos modernos de detecção de objetos. Neste relatório mostramos o processo de projetar e implementar uma paralelização em CPU de um extrator baseado em HOG, que chega a executar até XX.X vezes mais rápido que seu equivalente serial.*

1. Introdução

Em problemas de visão computacional, é comum surgirem situações onde é preciso descobrir se uma imagem contém uma ou mais instâncias de um objeto, seja esse uma pessoa, um cachorro, um carro, etc. Softwares que realizam esse tipo de detecção precisam primeiro transformar a imagem em um conjunto mais enxuto de dados, chamados *features*, que descrevem características chave da imagem ao mesmo tempo que tentam minimizar ruído e descartar informações irrelevantes. Uma vez extraído o vetor de features, a imagem pode ser classificada por meio de uma rede neural ou por métricas de distância a critério do programador.

Neste trabalho, foi desenvolvida uma versão paralelizada em CPU de um algoritmo extrator de features criado por Dalal & Triggs para detectar humanos, que resulta em um vetor de features chamado Histograma de Gradientes Orientados (em inglês, *Histogram of Oriented Gradients*, ou HOG). O programa foi desenvolvido em Java, e foi usada a ferramenta HPROF para o profiling e análise dos tempos de execução.

2. Histogram of Oriented Gradients

O descritor de features HOG, conforme apresentado por Dalal & Triggs em 2005, tem os seguintes passos:

1. Capturar uma janela de 64x128 pixels da imagem de entrada original.
2. Dividir a imagem da janela em 128 células de 8x8 pixels.
3. Calcular o vetor gradiente $G = [\Delta\alpha, \Delta\beta]$ de cada pixel da célula, onde $\Delta\alpha$ é a diferença entre os vizinhos inferior e superior, e $\Delta\beta$ a diferença entre os vizinhos da esquerda e da direita.

4. Montar um histograma de 9 colunas (10°, 30°, …, 170°) por célula, conforme os ângulos dos gradientes encontrados. A contribuição de cada gradiente é calculada distribuindo sua magnitude entre as duas colunas mais próximas por meio de uma interpolação linear.

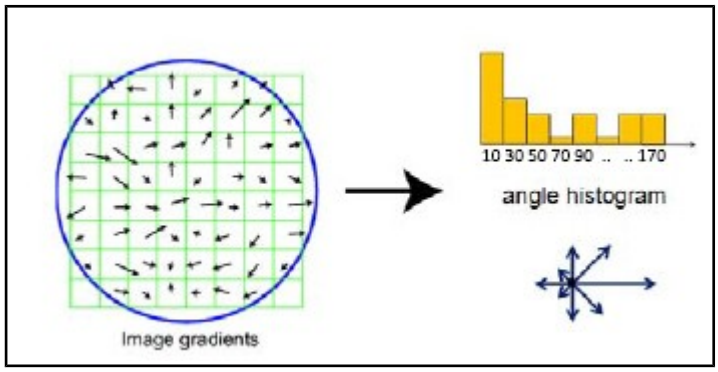

Figura 1. Construção do histograma de gradientes orientados

5. Agrupar as células em 105 blocos de 2x2 células com sobreposição (ou seja, blocos vizinhos compartilham 2 células) - na prática, cada bloco será a concatenação de 4 histogramas, resultando em vetores de 36 componentes.
6. Normalizar cada bloco a fim de remover possíveis ruídos provenientes de variações de contraste e iluminação.
7. Concatenar os vetores de cada bloco em um vetor final de 105 x 4 x 9 = 3,780 componentes, que descreve a janela toda.

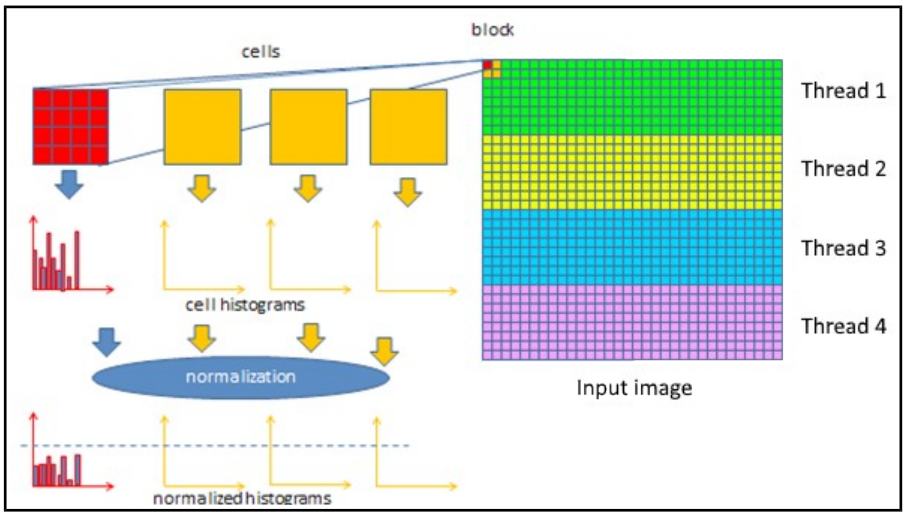

Figura 2. Visualização do processo de extração do HOG, já paralelizado

3. Paralelização do HOG

No método proposto por Dalal & Triggs, o passo 1 acima é necessário (juntamente com um pré-processamento da imagem) para detectar corretamente pessoas em escalas diferentes. Como o foco deste trabalho é apenas a construção do vetor descritor, e não a detecção de objetos, foi decidido que as imagens seriam sempre trabalhadas como um todo.

A paralelização do processo é relativamente simples, feita em duas etapas (mais detalhes sobre escolha dos trechos a serem paralelizados na seção 4). Primeiro a imagem de entrada é dividida em regiões onde cada thread irá operar. Assim, os histogramas das células 8x8 serão sempre escritos por apenas uma thread cada, sem concorrência.

Uma vez que todas as threads terminaram seus trabalhos e todas as células têm seus respectivos histogramas, é feito um novo paralelismo, também com uma divisão de carga, para a concatenação dos histogramas em blocos de 2x2 células e sua conseguinte normalização. Finalmente, quando todas as threads criaram os histogramas normalizados dos blocos, todos são concatenados em uma única saída.

4. Profiling

Para o profiling do código serial, foi utilizada a ferramenta HPROF, com o seguinte resultado (apenas as 10 primeiras entradas, por concisão):

```
CPU SAMPLES BEGIN (total = 217) Tue Jul  5 03:07:00 2016
rank   self   accum   count  trace method
 1    68.66% 68.66%     149 300343 java.lang.StrictMath.atan2
 2     6.45% 75.12%      14 300246 sun.java2d.cmm.lcms.LCMS.colorConvert
 3     4.15% 79.26%       9 300374 Hog.getOutput
 4     2.76% 82.03%       6 300342 Hog.getHistograms
 5     2.30% 84.33%       5 300372 com.sun.imageio.plugins.jpeg.JPEGImageWriter.writeImage
 6     1.38% 85.71%       3 300224 com.sun.imageio.plugins.jpeg.JPEGImageReader.readImage
 7     0.92% 86.64%       2 300094 java.lang.ClassLoader$NativeLibrary.load
 8     0.92% 87.56%       2 300221 sun.awt.image.ByteInterleavedRaster.putByteData
 9     0.92% 88.48%       2 300345 Hog.getHistograms
 10    0.92% 89.40%       2 300369 java.lang.Object.clone
```

Evidentemente, o processamento mais caro é o cálculo da arco-tangente durante a construção dos vetores gradientes, motivo pelo qual foi decidido dividir a carga de trabalho entre as threads. Em segundo lugar está a conversão da imagem para escala de cinza, que é inevitável. Em terceiro, a construção do vetor final de output, que é o segundo ponto onde consideramos necessária a paralelização. Em quarto lugar, a função de construção dos histogramas, que na realidade inclui o cálculo da arco-tangente e que portanto deveria estar em primeiro.

5. Resultados

Variando o tamanho da imagem e o número de threads, foram obtidas as seguintes relações de speedup em relação ao código serial:

Tabela 1. Dados utilizados para testes de eficiencia do algoritmo

Imagem	N. Threads	Speedup	Imagem	N. Threads	Speedup
Penguin.jpg (800x450)	2	2.285	Zebra.jpg (2133x1073)	2	4.244
	4	3.034		4	5.776
	8	3.008		8	7.104
	16	2.629		16	5.337
	32	2.595		32	5.695
Canguru.jpg (1280x854)	2	3.027	Tiger.jpg (2560x1600)	2	4.159
	4	3.636		4	5.976
	8	4.264		8	6.687
	16	4.460		16	6.682
	32	3.330		32	6.971

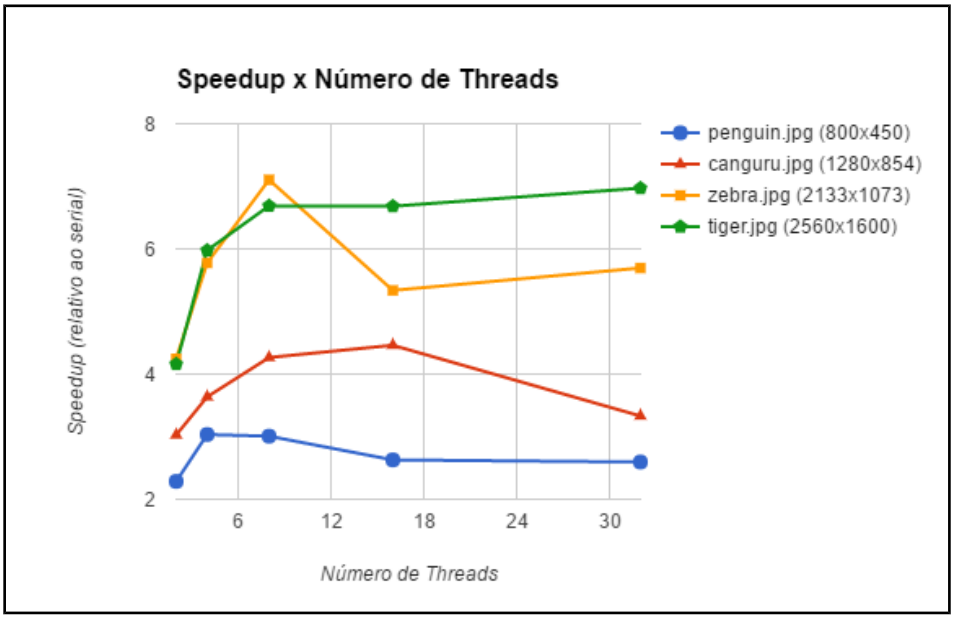

Figura 4. Gráfico de Speedup em relação ao algoritmo serial

6. Dificuldades Encontradas

Inicialmente, foi escolhida a linguagem Python para o desenvolvimento do projeto. Mais tarde, porém, descobrimos que a linguagem não oferece suporte como padrão a paralelismo, quando o objetivo é eficiência (o código em paralelo era muitas vezes mais lento que o serial). Mudamos então para a linguagem Java, a qual fornece um bom suporte ao paralelismo de algoritmos.

Ainda com o projeto Python em mente, foi planejado demonstrar o speedup do paralelismo em relação ao serialismo em tempo real, usando o feed de vídeo de uma webcam como input para o programa, e comparando o índice de Frames Por Segundo conforme a criação dos vetores gradientes; Essa ideia foi mais tarde descartada quando foi descoberto que usar Python era inviável, e que a complexidade de capturar vídeo em Java era muito acima do esperado.

Referências

Dalal, N. and Triggs, B., “Histograms of Oriented Gradients for Human Detection”, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA.

McCormick, C. “HOG Person Detector Tutorial. Disponível: http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/. Acesso: Junho/2016

Tomasi, C. “Histograms of Oriented Gradients”. Disponível: https://www.cs.duke.edu/courses/fall15/compsci527/notes/hog.pdf . Acesso: Junho/2016