Massively Parallel Multiclass Object Recognition

Helmut Sedding, Ferdinand Deger, Holger Dammertz, Jan Bouecke, Hendrik P. A. Lensch

Ulm University

Abstract

We present a massively parallel object recognition system based on a cortex-like structure. Due to its nature, this general, biologically motivated system can be parallelized efficiently on recent many-core graphics processing units (GPU). By implementing the entire pipeline on the GPU, by rigorously optimizing memory bandwidth and by minimizing branch divergence, we achieve significant speedup compared to both recent CPU as well as GPU implementations for reasonably sized feature dictionaries. We demonstrate an interactive application even on a less powerful laptop which is able to classify webcam images and to learn novel categories in real time.

Feature Extraction Inspired by Visual Cortex

We built our system closely along the approved base object recognition model of Mutch&Lowe [1].

Learning:

- *feature extraction* out of training images
- *feature weighting*: measure similarity of features to training images
- *SVM training* with obtained feature vector

Classification:

- *feature weighting*: measure similarity of features to test image
- *SVM classification* with obtained feature vector

Feature Vector Computation:

- *S-layers:* Convolution with templates (Gabor) or patches. Selectivity for a specific feature increases by applying a bell-shaped weighting curve to the resulting response.
- *C-layers:* non-linear reduction, namely maximum extraction over an area or an entire pyramid followed by sub-sampling. Reduces the amount of information and establishes scale and shift invariance.

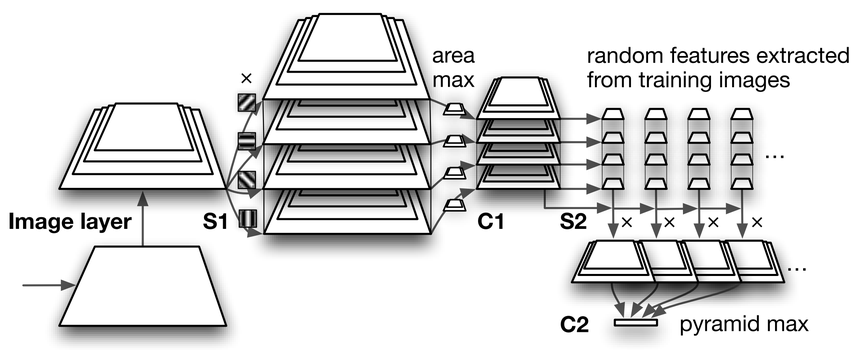


Figure: Hierarchical feature extraction and weighting pipeline.

Obs.: Layers operate mostly data-parallel ⇒ suits GPU computation.

Parallelization

GPU-Programming Principles:

- *SIMD:* Many threads doing the same operations
- *Caching:* Main memory access is expensive, requiring careful thread layout for fast throughput

Applying those principles we emphasize the following design decisions:

Single Thread per Output Pixel:

A thread reads a small area to interpolate over and writes one pixel. ⇒ exploits caching hardware effectively.

Layers S2 and C2: (most time critical)

We partition the problem such that each block of threads operates on one patch and performs the convolution scale by scale at all orientations. While threads operate in lock-step, blocks can run concurrently even if they do not execute the same code path. Therefore, each thread block can easily run differently sized loops, allowing for different patch size processing. We need p (= number of patches) blocks, and the algorithm scales with the number of processing units.
An additional, significant benefit is achieved by integrating the S2 and C2 layer into the same kernel.

Memory Structure

Important design decisions have to be made for the data structures and the organization of the data flow to reduce bandwidth utilization between CPU and GPU memory, and within the GPU. Some of them are:

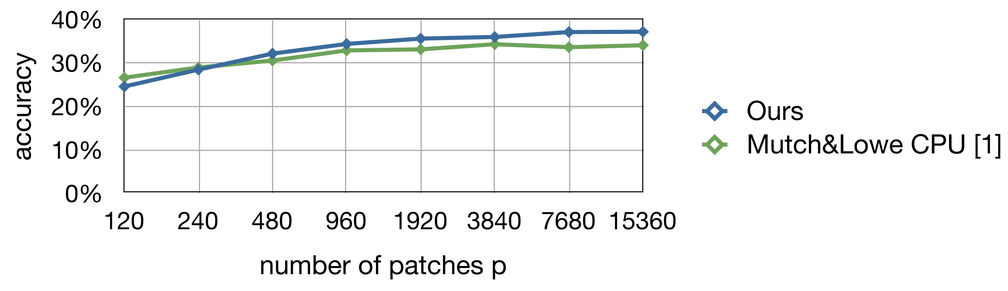
Main Memory to GPU Transfer: Reduced to a minimum by implementing the entire processing pipeline on the GPU.

Texture Cached Read-Only Access: In all layers, we used GPU texturing units which provide cached access to arrays (1D, 2D) and allow for automatic boundary handling.

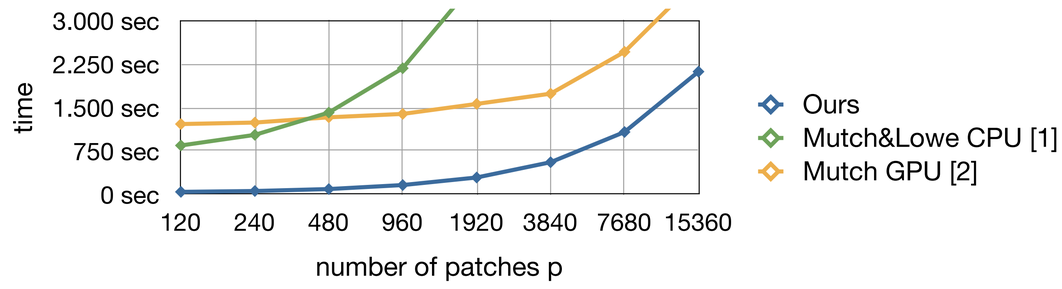
Image Pyramid Data Structure: Separate 2D textures for each pyramid and pyramid scale are tedious and inflexible to program in CUDA. We used a only one sliding 2D-texture over multiple kernel calls. Even with the kernel call overhead it is ≈ 50% faster.

Results

Correctness: We achieve classification rates comparable to Mutch&Lowe's reference implementation [1], here with Caltech-101:



Performance: Our GPU implementation in CUDA is vastly faster than the CPU reference implementation [1] and faster than Mutch's recent GPU cortex simulator framework (CNS) [2]:



Mobile Real Time Application The significant speedup, especially for moderate numbers of patches, allows us to address real-time object recognition even on small mobile GPUs such as a low end NVIDIA Geforce 9400M in a 2008' Apple Macbook Pro.

References

Jim Mutch and David G. Lowe.
Multiclass object recognition with sparse, localized features.
In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11–18, 2006.

Jim Mutch, Ulf Knoblich, and Tomaso Poggio.
CNS: a GPU-based framework for simulating cortically-organized networks.
Technical Report MIT-CSAIL-TR-2010-013 / CBCL-286, 2010.

helmut@sedding.net, ferdinand.deger@gmail.com, http://github.com/cumorc/cumorc