WORK-EFFICIENT PARALLEL NON-MAXIMUM SUPPRESSION FOR EMBEDDED GPU ARCHITECTURES

David Oro ⋆ *Carles Fernández* ⋆ *Xavier Martorell* † ‡ *Javier Hernando* †

⋆ Herta Security, Barcelona, Spain
† Universitat Politècnica de Catalunya, Barcelona, Spain
‡ Barcelona Supercomputing Center
Email: david.oro@hertasecurity.com

ABSTRACT

With the emergence of GPU computing, deep neural networks have become a widely used technique for advancing research in the field of image and speech processing. In the context of object and event detection, sliding-window classifiers require to choose the best among all positively discriminated candidate windows. In this paper, we introduce the first GPU-based non-maximum suppression (NMS) algorithm for embedded GPU architectures. The obtained results show that the proposed parallel algorithm reduces the NMS latency by a wide margin when compared to CPUs, even clocking the GPU at 50% of its maximum frequency on an NVIDIA Tegra K1. In this paper, we show results for object detection in images. The proposed technique is directly applicable to speech segmentation tasks such as speaker diarization.

Index Terms— Non-maximum suppression, deep neural networks, GPU computing, CUDA, object detection

1. INTRODUCTION

Recent advances in GPU computing performance have made the real-time execution of highly complex signal processing techniques a reality. Applications including advanced driver-assistance systems (ADAS), scene understanding, speech segmentation and speaker diarization, among others, leverage data-parallel GPU architectures. In these environments, embedded computing is playing an increasingly important role due to the power consumption constraints.

Typically, the most widely used object and event detection techniques rely on a sliding window approach, which yields multiple overlapping candidate windows with similarly high scores around the true location of the object. Non-maximum suppression (NMS) is the process of selecting a single representative candidate within this cluster of detections, so as to obtain a unique detection per object in the signal.

This work has been supported by the Spanish Ministry of Economy and Competitiveness under contracts TIN2012-34557, TIN2015-65316, PCIN-2013-067, the Generalitat de Catalunya under contracts 2014-SGR-1051, 2014-SGR-1660, and the European Commission under the Horizon 2020 program (RAPID project H2020-ICT-644312).

Deep neural networks (DNNs) have renewed the interest in applying fast NMS algorithms for object and event classification tasks. State-of-the-art DNN frameworks [1] have quickly emerged as a powerful machine learning tool that provides improved accuracy [2] albeit at a high computational training cost while involving huge amounts of data. Since DNNs are inherently data parallel, they are usually built and fine-tuned using high-end discrete GPUs. However, for the evaluation and deployment of such DNN models on real-world scenarios, embedded platforms featuring mobile GPUs such as the NVIDIA Tegra are quickly gaining traction.

Modern system-on-chip heterogeneous platforms feature low-power multicore ARM CPU cores combined with general-purpose GPUs. These embedded GPUs are quickly closing the performance gap with high-end discrete GPUs, and they are now powerful enough for handling massively parallel CUDA and OpenCL kernels.

In the field of speech segmentation it is also necessary to select the most appropriate windows localizing acoustic events in the temporal dimension, as they sample the acoustic recording timeline with small temporal increments and high overlapping. Additionally, window selection can also be applied in the context of feature extraction from 2D spectrogram images derived from speech tasks. Here, DNNs traditionally employed in computer vision are being increasingly used to automatically locate and extract features from spectrogram [3, 4, 5]. Hence, a GPU-based NMS algorithm would also benefit applications such as acoustic event detection, speaker diarization, and speech or speaker recognition.

Even though NMS is a required step for DNN models implemented as sliding-window classifiers [6, 7], they are still sequentially executed on CPUs and thus cannot exploit the vast amount of computing resources available on general-purpose embedded GPUs. For real-time speech processing and computer vision applications analyzing large amounts of simultaneous objects, a data-parallel GPU kernel that overcomes the latency constraints imposed by the data dependences of serial NMS implementations is thus required.

In this paper we present a fast and lightweight parallel ker-

nel implementation of NMS that targets the GPUs included in NVIDIA's Tegra K1 and X1 platforms. To the best of our knowledge, this is the first NMS implementation that fully exploit such GPU architectures, and it could be easily mapped to any multithreaded stream processor with minimal efforts.

2. NON-MAXIMUM SUPPRESSION

In the context of object detection, DNNs implemented as sliding-window classifiers require fusing multiple overlapping detections. As Figure 1 depicts, this fusion is an important step of such DNN-based object classification frameworks. Usually, the output of each detected window is a score derived from the last layer of the DNN. More particularly, this score represents a measure of the likelihood that the region enclosed by the window contains the object through which the DNN classifier has been trained. The score is thus degraded as the location and scale of the sliding window containing the object varies. As a result, the maximum score is obtained at the precise location and window dimensions, corresponding to the local maximum of the response function used by the DNN.

The goal of NMS is to extract a good, single representative from each set of clustered candidate object detections. Therefore, NMS resembles a classic clustering problem, and typically relies on two basic operations: (i) identifying the cluster to which each detection belongs, and (ii) finding a representative for each cluster.

Assuming rectangular bounding boxes, the positive output of a given binary DNN sliding-window classifier yields a tuple $\{x, y, w, h, s\}$, namely 2D coordinates (x, y), window width and height $w \times h$, and a score s for a detection $d \in D$, in which D is the set containing all detected objects. This NMS approach is usually implemented as a greedy iterative process, and involves defining a measure of similarity between windows while setting a threshold θ for window suppression.

Recent works [7, 8, 9] commonly rely on the abovementioned greedy NMS technique. These post-processing methods essentialy find the window with the maximum score, and then reject the remaining candidate windows if they have an intersection over union (IoU) larger than a learned threshold. However, no NMS latency benchmarks were disclosed in those works, and also parallelization remains unaddressed.

Another common NMS approach is to employ optimized versions of clustering algorithms, particularly *k-means* [10] or *mean shift* [11]. Unfortunately, *k-means* requires a predetermined number of clusters, which is unknown and difficult to estimate beforehand; and additionally only identifies convex clusters, so it cannot handle very non-linear data. On the other hand, *mean shift* is computationally intensive and often struggles with data outliers. Combining both methods may solve many of these problems in practice, but their iterative nature makes them difficult to parallelize and highly uncompetitive from a latency perspective.

Fig. 1. Visualization of the NMS process for a DNN-based face classifier. Pre-NMS (light boxes). Post-NMS (bold box).

A novel NMS proposal [12] based on the *affinity propagation* clustering algorithm overcomes the shortcomings derived from hard-coded thresholds of greedy NMS methods. However, this proposal is unworkable for real-time applications as the authors report a latency of 1000 ms to cluster 250 candidate windows.

3. PARALLEL IMPLEMENTATION

In order to exploit the underlying architecture of general-purpose embedded GPUs, an NMS kernel must expose a parallelization pattern in which each computing thread independently evaluates the overlapping between two given bounding boxes. The idea is to avoid at the maximum extent data dependences that serialize computations, and thus overcome the limitations in scalability derived from the traditionally iterative clustering process. Our proposal addresses this issue by adopting a *map/reduce* parallelization pattern which uses a boolean matrix both to encode candidate object detections and to compute their cluster representatives.

Figure 2 depicts a toy example of the proposed algorithm, in which an image frame contains three objects, three window clusters and nine detections. Our matrix encodes the relationship among all detections, initially assuming that all are possible cluster representatives (matrix of ones). Firstly, we decide that two windows d_i and d_j belong to the same cluster if their areas are overlapped beyond a given threshold; otherwise, a zero will be placed in the matrix coordinates (d_i, d_j) and (d_j, d_i). Secondly, we evaluate the non-zero values of each row, and again place zeroes if the row-indexed detection (d_i) is strictly smaller than the column-indexed one (d_j), thus discarding d_i as the cluster representative (grayed out in Figure 2). Finally, a horizontal AND reduction will preserve a single representative per cluster, thus completing the NMS.

Formalizing this process, let D be the set of detection windows and C the set of clusters for a given frame, with

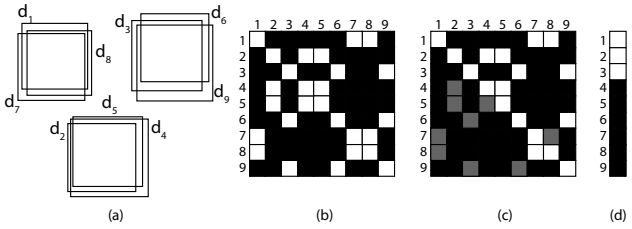

Fig. 2. Visualization of our GPU-NMS proposal: (a) example candidates generated by a detector (3 objects, 9 detections); boolean matrix after (b) clustering and (c) cancellation of non-representatives; and (d) result after AND reduction.

$C \subseteq D$. We build a boolean matrix B of size $D_{max} \times D_{max}$, being D_{max} an upper limit of the number of windows possibly generated by the detector at any frame. Let $A(\cdot)$ be an operator that returns the area of a window. Given an overlapping threshold $\theta \in [0, 1]$, two candidate windows d_i and d_j are assigned to the same cluster if $A(d_i \cap d_j)/A(d_i) \geq \theta$. Candidates within a cluster are discarded as representatives if $A(d_i) < A(d_j)$. The clipping process is performed in parallel independently for each detection $d \in D$ by a given computing thread. Since there are no data dependences among detections, this mapping strategy scales properly as the amount of GPU cores is increased.

The only required parameters for this algorithm are the overlapping threshold θ and the maximum number of possible candidates that can be generated by the detector D_{max}. Although this last constraint may initially seem to be an important drawback, in general conservative values for D_{max} turn to be very relaxed constraints. As an example, it is common that face detection models have a minimum face resolution of 24×24 pixels. In that case, the worst case scenario for a HD frame of 1920×1080 pixels would be a tiling of 80×45 faces of that size, yielding a matrix B of 3600^2 elements.

Internally, the *map* kernel (see Algorithm 1) must first compute the area a to effectively perform the overlapping test for each pair of detections $d_i, d_j \in D$. With the aim of preserving simplicity, we assume equal width and height for the bounding boxes. Therefore, each detection is redefined as a $\{x, y, z, s\}$ tuple, in which $z = w = h$.

Once the boolean matrix B has been computed, it is required to call a *reduce* kernel (see Algorithm 2) for selecting the optimal candidate from each row as it is depicted in Figure 2. This task is perfomed using AND operations in parallel for each row of B and can be implemented in a CUDA kernel by means of `__syncthreads_and(cond)`. This directive returns 1 only if the `cond` predicate evaluates to true for all threads of the CUDA block, and is directly translated to the hardware-accelerated `BAR.RED.AND` assembly instruction. Therefore, it is possible to split the AND reductions of B by creating D_{max}/k partitions and then assigning each partition to a given thread block. Under this parallelization

Algorithm 1: MAPKERNEL

Data: Matrix B and vector D
begin
  $i \leftarrow$ `blockIdx.x * blockDim.x + threadIdx.x`
  $j \leftarrow$ `blockIdx.y * blockDim.y + threadIdx.y`
  if $D[i].s < D[j].s$ **then**
    $a \leftarrow (D[j].z + 1) * (D[j].z + 1)$
    $w \leftarrow max(0, min(D[i].x + D[i].z), D[j].x + D[j].z)$
      $-max(D[i].x, D[j].x) + 1)$
    $h \leftarrow max(0, min(D[i].y + D[i].z), D[j].y + D[j].z)$
      $-max(D[i].y, D[j].y) + 1)$
    $B[i * D_{max} + j] \leftarrow (\frac{w*h}{a} < \theta) \wedge D[j].z \neq 0$
  end
end

Algorithm 2: REDUCEKERNEL

Data: Matrix D, value k, and vector V of size D_{max}
begin
  $i \leftarrow$ `blockIdx.x`
  $j \leftarrow i * D_{max} +$ `threadIdx.x`
  $n \leftarrow D_{max}/k$
  $V[i] \leftarrow$ `__syncthreads_and`$(B[j])$
  for 1 **to** $k - 1$ **do**
    $j \leftarrow j + n$
    $V[i] \leftarrow$ `__syncthreads_and`$(V[i]$ && $B[j])$
  end
end

pattern, each thread is synchronized, and simultaneously reduces the boolean values stored in the partition. Parameter k is experimentally determined so that the GPU achieves the highest occupancy. Since we are dealing with a square matrix, it is required to call k times the reduction directive within the kernel assuming a CUDA block of size D_{max}/k and a grid size equal to the size of the input set detections D. A vector V of size D_{max} is required for temporarily storing partial reductions of CUDA blocks.

4. EXPERIMENTAL RESULTS

In order to evaluate the latency of the proposed GPU-based NMS method, we conducted several experiments under different scenarios. The selected input consisted of a video from the *SVT HD multi-format test set*[1] featuring approximately 60 simultaneous faces per frame (named `crowd_run`), and the *83rd Academy Awards Nominees*[2] image with 147 faces. As a result of this, the output of a DNN-based face classifier was used as the input of the proposed NMS algorithm. The tested

[1] `ftp://vqeg.its.bldrdoc.gov/HDTV/`
[2] `http://tinyurl.com/o5n97ra/`

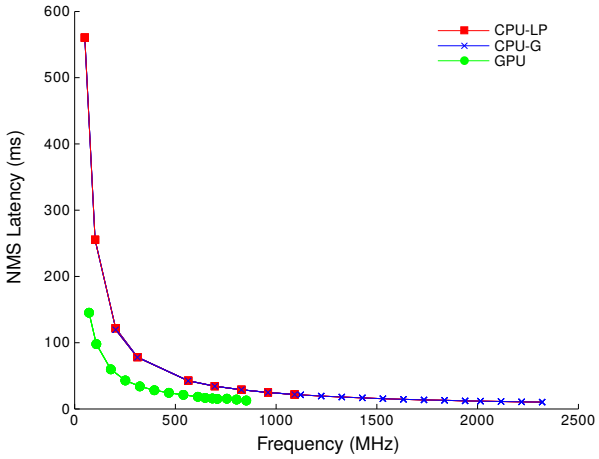

Fig. 3. NMS latency on GPU, CPU-LP and CPU-G cores for the selected input image.

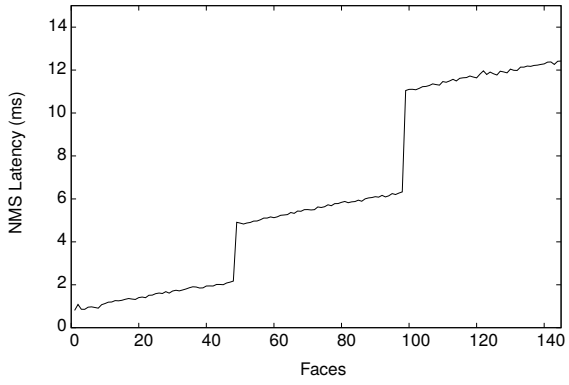

Fig. 4. NMS latency on GPU for the input image (*k=4*).

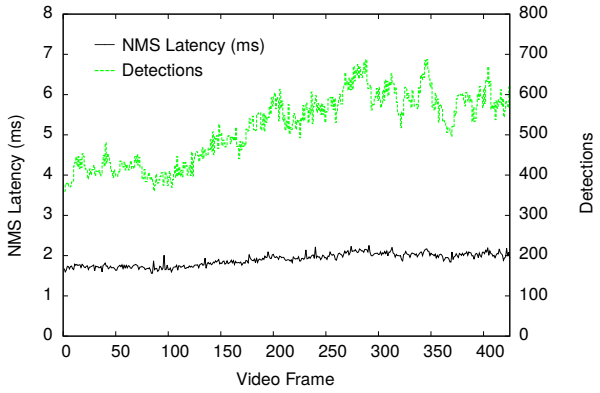

Fig. 5. NMS latency on GPU for the input video (*k=4*).

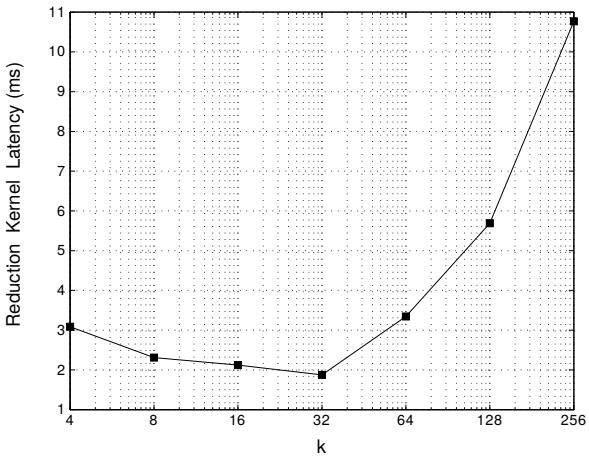

Fig. 6. Reduction kernel latency depending on *k* value.

platform was a Jetson TK1 board equipped with a Tegra K1 chip, and flashed with the JetPack TK1 v1.2 image.

Since the Tegra K1 chip features 4 CPU cores plus a fifth low-power *companion core*, we compared the latency of the GPU-based NMS against a high-performing core (CPU-G) and the low-power core (CPU-LP) running OpenCV's $\mathcal{O}(n^2)$ NMS as in [13]. Due to the employed DVFS hardware techniques, the frequency of the GPU varies between 72 MHz and 850 MHz, whereas the CPU-LP core ranges 51 MHz - 1 GHz, and a given CPU-G core 204 MHz - 2.3 GHz. Figure 3 shows that a GPU clocked at 50% of its maximum frequency outperforms both CPU types also when they are operating at 50% of its maximum frequency. It should be noted that for battery-powered fanless solutions the chip must be underclocked.

In general, the GPU-based NMS algorithm also scales properly as the number of simultaneous faces and detections increase. Figure 4 shows that the latency spikes at 49 and 99 simultaneous faces (1024 and 2048 detections), respectively. These results are in line with Algorithm 2, since *k=4* yields a 1024-thread block. Analogously, Figure 5 shows a quasi-constant GPU latency due to fact that the amount of simultaneous detections of a given frame is less than 1024.

In line with the obtained results, it is thus possible to explore multiple thread block partitions for the reduction kernel by varying the k value. Figure 6 shows that the lowest latency for the reduction kernel is achieved when $k = 32$ using the same input image featuring 147 faces (corresponding to 2997 detections).

5. CONCLUSIONS

In this paper, we have presented efficient NMS kernels for embedded GPUs based on the Tegra K1 chip. The obtained results show that the GPU generally beats CPU cores even clocking the GPU at a 50% of its maximum frequency. In future works, we plan to profile the proposed kernels on chips with higher GPU core counts such as Tegra X1.

6. REFERENCES

[1] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*. ACM, 2014, pp. 675–678.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 8614–8618.

[4] Ossama Abdel-Hamid, Li Deng, and Dong Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *Proceedings of the 14th Annual Conference of the International Speech Communication Association*, 2013, pp. 3366–3370.

[5] Tara N Sainath, Brian Kingsbury, George Saon, Hagen Soltau, Abdel-rahman Mohamed, George Dahl, and Bhuvana Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39–48, 2015.

[6] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, and Dave Ferguson, "Real-time pedestrian detection with deep network cascades," in *Proceedings of the British Machine Vision Conference*, 2015.

[7] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5325–5334.

[8] Sachin Sudhakar Farfade, Mohammad Saberian, and Li-Jia Li, "Multi-view face detection using deep convolutional neural networks," in *arXiv preprint arXiv:1502.02766*, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[10] S.A. Arul Shalom, Manoranjan Dash, and Minh Tue, "Efficient k-means clustering using accelerated graphics processors," in *Data Warehousing and Knowledge Discovery*, pp. 166–175. Springer, 2008.

[11] Peihua Li and Lijuan Xiao, "Mean shift parallel tracking on GPU," in *Pattern Recognition and Image Analysis*, pp. 120–127. Springer, 2009.

[12] Rasmus Rothe, Matthieu Guillaumin, and Luc Van Gool, "Non-maximum suppression for object detection by passing messages between windows," in *Computer Vision – ACCV 2014*, vol. 9003 of *Lecture Notes in Computer Science*, pp. 290–306. Springer, 2015.

[13] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.