NVIDIA® cuDNN R2 Release Notes

Release Candidate 1 – December 7, 2014

What's New In cuDNN R2

- Performance of many routines – especially forward convolutions – have been improved considerably.
- Forward convolution is now implemented via several different algorithms, and the interface allows the application to choose one of these algorithms specifically or to specify a strategy (e.g., prefer fastest, use no additional working space) by which the library should select the best algorithm automatically. The four algorithms currently given are as follows:
    - IMPLICIT_GEMM corresponds to the sole algorithm that was provided in cuDNN R2; it is the only algorithm that supports all input sizes while using no additional working space.
    - IMPLICIT_PRECOMP_GEMM is a modification of this approach that uses a small amount of working space (C*R*S*sizeof(int) bytes) to achieve significantly higher performance. As such, the "prefer fastest" strategy will almost always select this algorithm. This algorithm achieves its highest performance when zero-padding is not used.
    - GEMM is a BETA preview of an "im2col"-based approach followed by an otherwise-pure matrix multiplication that obeys cuDNN's input and output stridings, avoiding explicit transpositions of the input or output. Note that this algorithm requires significant working space, and it is not yet tuned for optimal performance.
    - DIRECT is a placeholder for a future implementation of direct convolution.
- The interface of cuDNN has been generalized so that data sets with other than two spatial dimensions (e.g., 1D or 3D data) can be supported in future releases.
    - Note: while the interface now allows arbitrary N-dimensional tensors, most routines in this release remain limited to two spatial dimensions in cuDNN R2. This may be relaxed in future releases based on community feedback.
    - As a BETA preview in this release, the convolution forward, convolution weight and data gradient, and cudnnSetTensor/cudnnScaleTensor routines now support 3D datasets through the "Nd" interface. Note, however, that these 3D routines have not yet been tuned for optimal performance. This will be improved in future releases.
- Many routines now allow alpha and beta scaling parameters of the following form:

    $$C := \alpha * OP(...) + \beta * C$$

    This replaces the cudnnAccumulateResult_t enumerated type in cuDNN R1, which allowed only two combinations, namely (alpha, beta) = (1.0, 0.0) and (alpha, beta) = (1.0, 1.0).
- The pooling routines now allow zero-padding of the borders in a manner similar to what was already supported for convolutions.
- OS X is now supported.

- Support for arbitrary strides is improved. While performance is still generally best tuned for cases where the data is in NCHW order and tightly packed (i.e., the stride of especially innermost dimension is 1), arbitrary tensor orderings and/or stridings are supported and performance of these has been improved in some routines. Further improvements may be made in future releases based on community feedback.
    - cudnnSetTensor4dDescriptor now supports CUDNN_TENSOR_NHWC.
    - The performance of the cudnnSoftmax*() routines when using CUDNN_SOFTMAX_MODE_CHANNEL with tensor layouts other than ***C has been improved considerably.
    - Note: While tensors allow arbitrary data layout, filters are still assumed to be stored in KCRS order and tightly packed.

Issues Resolved

- When NULL pointers were passed to the convolution routines, CUDNN_STATUS_MAPPING_ERROR (which is normally indicative of an internal error) could be returned in some circumstances. These routines now check for NULL arguments and will return CUDNN_STATUS_BAD_PARAM if one is found.
- The static library build for Windows has been removed due to linking issues across versions of Visual Studio.
- Some routines could fail unexpectedly if certain dimensions (typically N or C) were very large. All routines now either handle these large dimensions or return CUDNN_STATUS_NOT_SUPPORTED.
- Activation functions allow in-place operation (i.e., the input and output pointers and the tensors structures describing them are identical). This is now documented, and a check that input and output strides are equal if the input and output pointers are equal has been added.

Known Issues

- If some dimension of a tensor is 1, checks for unsupported (e.g., zero) strides could be relaxed.
- Max pooling may return the wrong result for certain input tensor sizes.
- Documentation of the exact function implemented by non-trivial routines should be explicit.