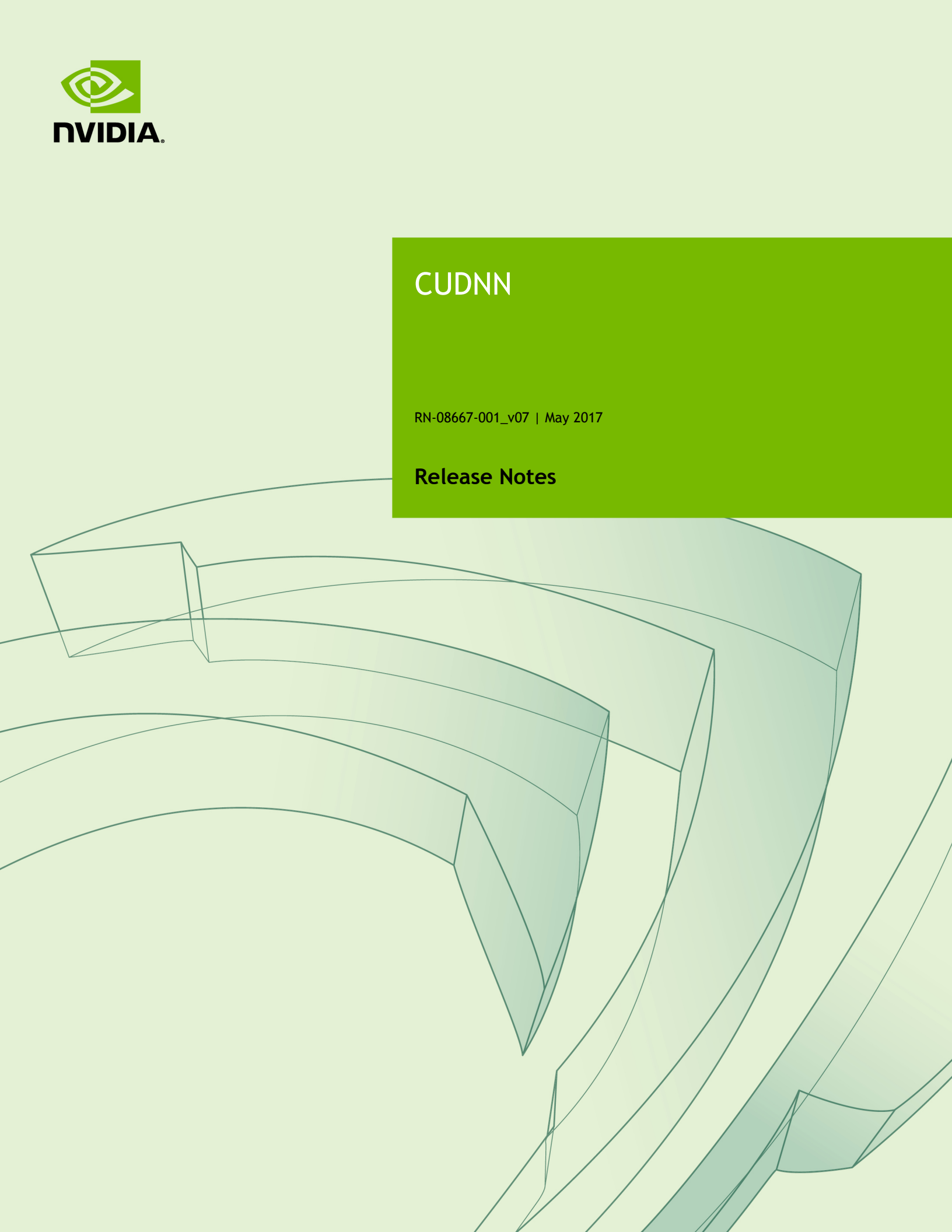

CUDNN

RN-08667-001_v07 | May 2017

Release Notes

TABLE OF CONTENTS

Chapter 1. CUDNN OVERVIEW

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks. It provides highly tuned implementations of routines arising frequently in DNN applications:

- Convolution forward and backward, including cross-correlation
- Pooling forward and backward
- Softmax forward and backward
- Neuron activations forward and backward:
  - Rectified linear (ReLU)
  - Sigmoid
  - Hyperbolic tangent (TANH)
- Tensor transformation functions
- LRN, LCN and batch normalization forward and backward

cuDNN's convolution routines aim for performance competitive with the fastest GEMM (matrix multiply) based implementations of such routines while using significantly less memory.

cuDNN features customizable data layouts, supporting flexible dimension ordering, striding, and subregions for the 4D tensors used as inputs and outputs to all of its routines. This flexibility allows easy integration into any neural network implementation and avoids the input/output transposition steps sometimes necessary with GEMM-based convolutions.

cuDNN offers a context-based API that allows for easy multi-threading and (optional) interoperability with CUDA streams.

Chapter 2. CUDNN RELEASE NOTES V7.0.3

Key Features and Enhancements

Performance improvements for various cases:

- Forward Grouped Convolutions where input channel per groups is 1, 2 or 4 and hardware is Volta or Pascal.
- **`cudnnTransformTensor()`** where input and output tensor is packed.

This is an improved fallback, improvements will not be seen in all cases.

Known Issues

The following are known issues in this release:

- **`CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING`** may cause **`CUDA_ERROR_ILLEGAL_ADDRESS`**. This issue affects input images of just one 1 pixel in width and certain **`n`**, **`c`**, **`k`**, **`h`** combinations.

Fixed Issues

The following issues have been fixed in this release:

- **`AddTensor`** and **`TensorOp`** produce incorrect results for half and INT8 inputs for various use cases.
- **`cudnnPoolingBackward()`** can produce incorrect values for rare cases of non-deterministic MAX pooling with **`window_width > 256`**. These rare cases are when the maximum element in a window is duplicated horizontally (along width) by a stride of **`256*k`** for some **`k`**. The behavior is now fixed to accumulate derivatives for the duplicate that is left-most.
- **`cudnnGetConvolutionForwardWorkspaceSize()`** produces incorrect workspace size for algorithm **`FFT_TILING`** for 1d convolutions. This only occurs for large sized

convolutions where intermediate calculations produce values greater than 2^31 (2 to the power of 31).

- **`CUDNN_STATUS_NOT_SUPPORTED`** returned by **`cudnnPooling*()`** functions for small **`x`** image (**`channels * height * width < 4`**).

Chapter 3. CUDNN RELEASE NOTES V7.0.2

Key Features and Enhancements

This is a patch release of cuDNN 7.0 and includes bug fixes and performance improvements mainly on Volta.

Algo 1 Convolutions Performance Improvements

Performance improvements were made to **`CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM`**, **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1`**, and **`CUDNN_CONVOLUTION_BWD_DATA_ALGO_1`**. These improvements consist of new SASS kernels and improved heuristics. The new kernels implement convolutions over various data sizes and tile sizes. The improved heuristics take advantage of these new kernels.

Known Issues

The following are known issues in this release:

- **`cudnnGetConvolutionForwardWorkspaceSize()`** returns overflowed size_t value for certain input shape for **`CUDNN_CONVOLUTION_*_ALGO_FFT_TILING`**.

Fixed Issues

The following issues have been fixed in this release:

- Batch Norm **`CUDNN_BATCHNORM_SPATIAL_PERSISTENT`** might get into race conditions in certain scenarios.
- cuDNN convolution layers using **`TENSOR_OP_MATH`** with fp16 inputs and outputs and fp32 compute will use “round to nearest” mode instead of “round to zero” mode as in 7.0.1. This rounding mode has proven to achieve better results in training.

- Fixed synchronization logic in the **CUDNN_CTC_LOSS_ALGO_DETERMINISTIC** algo for CTC. The original code would hang in rare cases.
- Convolution algorithms using **TENSOR_OP_MATH** returned a workspace size from ***GetWorkspaceSize()** smaller than actually necessary.
- cuDNN pooling backwards fails for pooling window size > 256.
- The results of int8 are inaccurate in certain cases when calling **cudnnConvolutionForward()** in convolution layer.
- **cudnnConvolutionForward()** called with **xDesc's channel = yDesc's channel = groupCount** could compute incorrect values when vertical padding > 0.

Chapter 4.
CUDNN RELEASE NOTES V7.0.1

cuDNN v7.0.1 is the first release to support the Volta GPU architecture. In addition, cuDNN v7.0.1 brings new layers, grouped convolutions, and improved convolution find as error query mechanism.

Key Features and Enhancements

This cuDNN release includes the following key features and enhancements.

Tensor Cores
Version 7.0.1 of cuDNN is the first to support the Tensor Core operations in its implementation. Tensor Cores provide highly optimized matrix multiplication building blocks that do not have an equivalent numerical behavior in the traditional instructions, therefore, its numerical behavior is slightly different.

`cudnnSetConvolutionMathType`, `cudnnSetRNNMatrixMathType`, and `cudnnMathType_t`
The **`cudnnSetConvolutionMathType`** and **`cudnnSetRNNMatrixMathType`** functions enable you to choose whether or not to use Tensor Core operations in the convolution and RNN layers respectively by setting the math mode to either **`CUDNN_TENSOR_OP_MATH`** or **`CUDNN_DEFAULT_MATH`**.

Tensor Core operations perform parallel floating point accumulation of multiple floating point products.

Setting the math mode to **`CUDNN_TENSOR_OP_MATH`** indicates that the library will use Tensor Core operations.

The default is **`CUDNN_DEFAULT_MATH`**. This default indicates that the Tensor Core operations will be avoided by the library. The default mode is a serialized operation

whereas, the Tensor Core is a parallelized operation, therefore, the two might result in slightly different numerical results due to the different sequencing of operations.

The library falls back to the default math mode when Tensor Core operations are not supported or not permitted.

cudnnSetConvolutionGroupCount
A new interface that allows applications to perform convolution groups in the convolution layers in a single API call.

cudnnCTCLoss
cudnnCTCLoss provides a GPU implementation of the Connectionist Temporal Classification (CTC) loss function for RNNs. The CTC loss function is used for phoneme recognition in speech and handwriting recognition.

CUDNN_BATCHNORM_SPATIAL_PERSISTENT
The **CUDNN_BATCHNORM_SPATIAL_PERSISTENT** function is a new batch normalization mode for **cudnnBatchNormalizationForwardTraining** and **cudnnBatchNormalizationBackward**. This mode is similar to **CUDNN_BATCHNORM_SPATIAL**, however, it can be faster for some tasks.

cudnnQueryRuntimeError
The **cudnnQueryRuntimeError** function reports error codes written by GPU kernels when executing **cudnnBatchNormalizationForwardTraining** and **cudnnBatchNormalizationBackward** with the **CUDNN_BATCHNORM_SPATIAL_PERSISTENT** mode.

cudnnGetConvolutionForwardAlgorithm_v7
This new API returns all algorithms sorted by expected performance (using internal heuristics). These algorithms are output similarly to **cudnnFindConvolutionForwardAlgorithm**.

cudnnGetConvolutionBackwardDataAlgorithm_v7
This new API returns all algorithms sorted by expected performance (using internal heuristics). These algorithms are output similarly to **cudnnFindConvolutionBackwardAlgorithm**.

cudnnGetConvolutionBackwardFilterAlgorithm_v7
This new API returns all algorithms sorted by expected performance (using internal heuristics). These algorithms are output similarly to **cudnnFindConvolutionBackwardFilterAlgorithm**.

CUDNN_REDUCE_TENSOR_MUL_NO_ZEROS
The **MUL_NO_ZEROS** function is a multiplication reduction that ignores zeros in the data.

CUDNN_OP_TENSOR_NOT
The **OP_TENSOR_NOT** function is a unary operation that takes the negative of (alpha*A).

cudnnGetDropoutDescriptor
The **cudnnGetDropoutDescriptor** function allows applications to get dropout values.

Using cuDNN v7.0.1

Ensure you are familiar with the following notes when using this release.

- Multi-threading behavior has been modified. Multi-threading is allowed only when using different cuDNN handles in different threads.
- In **cudnnConvolutionBackwardFilter**, dilated convolution did not support cases where the product of all filter dimensions was odd for half precision floating point. These are now supported by **CUDNN_CONVOLUTION_BWD_FILTER_ALGO1**.
- Fixed bug that produced a silent computation error for when a batch size was larger than 65536 for **CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM**.
- In **getConvolutionForwardAlgorithm**, an error was not correctly reported in v5 when the output size was larger than expected. In v6 the **CUDNN_STATUS_NOT_SUPPORTED**, error message displayed. In v7, this error is modified to **CUDNN_STATUS_BAD_PARAM**.
- In **cudnnConvolutionBackwardFilter**, cuDNN now runs some exceptional cases correctly where it previously erroneously returned **CUDNN_STATUS_NOT_SUPPORTED**. This impacted the algorithms **CUDNN_CONVOLUTION_BWD_FILTER_ALGO0** and **CUDNN_CONVOLUTION_BWD_FILTER_ALGO3**.

Deprecated Features

The following routines have been removed:

- **cudnnSetConvolution2dDescriptor_v4**
- **cudnnSetConvolution2dDescriptor_v5**
- **cudnnGetConvolution2dDescriptor_v4**
- **cudnnGetConvolution2dDescriptor_v5**

Only the non-suffixed versions of these routines remain.

The following routines have been created and have the same API prototype as their non-suffixed equivalent from cuDNN v6:

- **`cudnnSetRNNDescriptor_v5`** - The non-suffixed version of the routines in cuDNN v7.0.1 are now mapped to their **`_v6`** equivalent.

Attention It is strongly advised to use the non-suffixed version as the **`_v5`** and **`_v6`** routines will be removed in the next cuDNN release.

- **`cudnnGetConvolutionForwardAlgorithm`**, **`cudnnGetConvolutionBackwardDataAlgorithm`**, and **`cudnnGetConvolutionBackwardFilterAlgorithm`** - A **`_v7`** version of this routine has been created. For more information, see the *Backward compatibility and deprecation policy* chapter of the cuDNN documentation for details.

Known Issues

- cuDNN pooling backwards fails for pooling window size > 256.

Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, cuFFT, cuSPARSE, DIGITS, DGX, DGX-1, Jetson, Kepler, NVIDIA Maxwell, NCCL, NVLink, Pascal, Tegra, TensorRT, and Tesla are trademarks and/or registered trademarks of NVIDIA Corporation in the Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright