CULA Programmer's Guide
www.culatools.com

Release 1.1

EM Photonics, Inc.
www.emphotonics.com

November 25, 2009

CONTENTS

INTRODUCTION

This document describes CULA™, an implementation of the Linear Algebra PACKage (LAPACK) interface for *CUDA*™-enabled NVIDIA® graphics processing units (GPUs).

CULA is a next-generation linear algebra package that uses the GPU as a co-processor to achieve speedups over existing linear algebra packages. CULA provides the same functionality you receive with your existing package, only at a greater speed.

CULA provides easy access to the NVIDIA computing resources available in your computer system. The library is a self-contained package that enhances linear algebra programs with little to no knowledge of the GPU computing model. CULA presents four distinct interfaces:

- **Standard** - The functions in this interface work on data in main memory, as in the Netlib LAPACK interface, but with semantics more familiar to the C and C++ programmer.

- **Device** - This interface follows the standards set forth in the NVIDIA *CUBLAS* package. In this interface, the user allocates and populates GPU memory and then calls CULA functions to operate on that memory. This is useful for closed-loop GPU systems that do large sections of processing on the GPU.

- **Fortran** - This functions of this interface match the **Standard** interface but use Fortran calling conventions. The user needs no knowledge of GPU computing, and will feel comfortable if already familiar with LAPACK.

- **Bridge** - This interface is targeted at users who are porting existing linear algebra codes to a GPU-accelerated environment. This interface provides a migration path by matching the function names and signatures of several popular linear algebra packages. It additionally provides a fallback to CPU execution when a user does not have a GPU or when problem size is too small to take advantage of GPU execution.

For maximum compatibility with the conventions of existing linear algebra packages, CULA uses column-major data storage and 1-based indexing, even on our C-oriented interfaces. More details are available in the *Programming Considerations* chapter.

1.1 Versions

The official release of CULA is available in several forms.

- **Basic** - The basic package contains a basic set of functions and is free of cost. This version supports single-precision real and complex data types.

- **Premium** - The premium package is a low-cost solution that provides a greater set of functionality than the **Basic** package and includes all data precisions. This package is intended for institutional and academic use.

- **Commercial** - The commercial package contains the functionality of the premium package, but permits the distribution of CULA-powered applications.

The following table summarizes the features of these three versions:

	Price	Precisions	Functions	Redistributable	Support
Basic	Free	S, C	6	Yes	Public forums
Premium	See website	S, D, C, Z	Many (see below)	For internal use	Ticket system
Commercial	See website	S, D, C, Z	Many	Yes	Email & phone

1.1.1 Available Functions

CULA implements the most frequently requested functions from LAPACK. As CULA evolves, more functions and function variants will be supported.

CULA Basic contains the following LAPACK function equivalents:

Function	Computation Routine
gels	Least squares system solve
geqrf	QR factorization
gesv	System solve
gesvd	Singular value decomposition
getrf	LU decomposition
gglse	Constrained least squares system solve

CULA Premium and Commercial support the following functions:

Function	Computation Routine
bdsqr	Singular value generation (Bidiagonal)
gebrd	Bidiagonalization
gelqf	LQ factorization
gels	Least squares system solve
geqrf	QR factorization
gerqf	RQ factorization
gesv	System solve
gesvd	Singular value decomposition
getrf	LU decomposition
getri	Matrix inverse
getrs	LU backsolve
gglse	Constrained least squares system solve
ggrqf	Orthogonal matrix factorize (RQ)
orgbr	Orthogonal matrix generate (Bidiagonal)
orglq	Orthonormal matrix generate (LQ)
orgqr	Orthonormal matrix generate (QR)
ormlq	Orthogonal matrix multiply (LQ)
ormqr	Orthogonal matrix multiply (QR)
ormrq	Orthogonal matrix multiply (RQ)
posv	Symmetric positive definite system solve
potrf	Cholesky decomposition
potrs	Cholesky backsolve
trtri	Triangular matrix inverse
trtrs	Triangular solve
ungbr	Unitary matrix generate (Bidiagonal)
unglq	Unitary matrix generate (LQ)
ungqr	Unitary matrix generate (QR)
unmlq	Unitary matrix multiply (LQ)
unmqr	Unitary matrix multiply (QR)
unmrq	Unitary matrix multiply (RQ)

1.2 Supported Operating Systems

CULA intends to support the full range of operating systems that are supported by *CUDA*. Installers are currently available for Windows and Linux in 32-bit and 64-bit versions. A 32-bit Mac OS X installer is also available. CULA has been tested on the following systems:

- Windows XP 32 / 64

- Windows Vista 32 / 64

- Ubuntu Linux 9.04

- Red Hat Enterprise Linux 5.3 / 4.7

- Fedora 11

- Mac OS X 10.5 (Leopard) - single precision only. Please see the release notes.

Please provide feedback on any other systems on which you attempt to use CULA. Although we are continually testing CULA on other systems, at present we officially support the above list. If your system is not listed, please let us know through the provided feedback channels.

1.3 Attributions

This work has been made possible by the NASA Small Business Innovation Research (SBIR) program. We recognize NVIDIA for their support.

CULA is built on NVIDIA CUDA 2.3 and NVIDIA *CUBLAS*.

CULA uses the Intel® Math Kernel Library (MKL) internally. For more information, please see the MKL product page at http://www.intel.com/software/products/mkl.

The original version of LAPACK from which CULA implements a similar interface can be obtained at http://www.netlib.org/lapack.

1.4 Notations

This document uses the following notational formats:

- Italics is used for names: *CULA*.

- Fixed-width is used for code: `code`.

LICENSE

EM Photonics, Inc.

Software License Agreement

CULA(TM) BASIC or CULA PREMIUM

IMPORTANT - READ BEFORE COPYING, INSTALLING, USING OR DISTRIBUTING

This is a legal agreement ("Agreement") between either a Subscriber or a Licensee, the party licensing either CULA Basic or CULA Premium, and EM PHOTONICS Inc., a Delaware corporation with its principal place of business at 51 East Main Street, Newark, Delaware, 19711 ("Licensor"). BY CLICKING ON THE "AGREE" BUTTON BELOW AND PRESSING THE ENTER KEY, YOU ACKNOWLEDGE THAT YOU HAVE READ ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT, UNDERSTAND THEM, AND AGREE TO BE LEGALLY BOUND BY THEM. If you do not agree with the terms of this Agreement, you may not download, install, use or distribute either CULA Basic or CULA Premium, as applicable.

1 SCOPE; DEFINITIONS. This Agreement governs your rights and obligations surrounding the permitted download, installation, use and, if applicable, distribution of either CULA Basic or CULA Premium. Unless otherwise defined in this Section 1, the capitalized terms used in this Agreement shall be defined in the context in which they are used. The following terms shall have the following meanings:

1.1 "Commercial Purpose" means the use, reproduction or distribution, directly or indirectly, of the Software, or any portion of the foregoing, that is intended to result in a direct or indirect pecuniary gain or any other consideration or economic benefit to any person or entity involved in such use, reproduction or distribution. Examples of a Commercial Purpose, include without limitation, (v) integrating the Software with other software or hardware for sale, (w) licensing the Software for a fee, (x) using the Software to provide a service to a third party, (y) selling the Software, or (z) distributing the Software for use with other products or other services.

1.2 "CULA Basic" means Licensor's limited, pre-complied implementation of linear algebra routines for certain third party graphics processing units.

1.3 "CULA Premium" means Licensor's more expanded, pre-compiled implementation of linear algebra routines for certain third party graphic processing units.

1.4 "Licensee" shall mean an individual or entity who has registered as a Licensee on www.culatools.com/register to use CULA Premium and who has paid the applicable license fees for such use.

1.5 "Intellectual Property Rights" shall mean all proprietary rights, including all patents, trademarks, copyrights, know-how, trade secrets, mask works, including all applications and registrations thereto, and any other similar protected rights in any country.

1.6 "Software" means collectively the CULA Basic and the CULA Premium.

1.7 "Subscriber" shall mean an individual or entity who has registered as a subscriber on www.culatools.com/register to use CULA Basic.

2 GRANT OF LICENSE.

2.1 CULA Basic License Grant. Only a Subscriber may exercise the rights under this Section 2.1. Subject to the terms and conditions of this Agreement, Licensor hereby grants to such Subscriber a world-wide, non-transferable, non sub-licensable, non-exclusive, perpetual license to do any of the following with respect to CULA Basic in each case, in accordance with the Documentation and the system minimum user requirements:

1. download, install and use CULA Basic for any purpose;

2. distribute certain unmodified CULA Basic files identified on Exhibit A (the "Distributable Files") on a limited stand alone basis to any third party;

3. download and use the Documentation and

4. reproduce CULA Basic and/or the Documentation as strictly necessary in exercising its rights under this Section 2.1.

2.2 CULA Premium License Grant. Subject to the terms and conditions of this Agreement, Licensor hereby grants each Licensee a world-wide, non-transferable, non sub-licensable, non-exclusive, perpetual license to do any of the following with respect to CULA Premium:

1. download, install and use CULA Premium only for internal use;

2. reproduce CULA Premium and/or the Documentation as strictly necessary in exercising its rights under this Section 2.2.

2.3 Limitations on License; Restricted Activities. Each of the Subscribers and/or Licensees recognize and agree that the Software is the property of Licensor, contain valuable assets and proprietary information and property of Licensor, and are provided to such Subscriber or Licensee, as the case may be, under the terms and conditions of this Agreement. Notwithstanding anything to the contrary in this Agreement, each Subscriber and/or Licensee agrees that he, she or it shall not do any of the following without Licensor's prior written consent:

1. Download, use, install, deploy, perform, modify, license, display, reproduce, distribute or disclose the Software other than as allowed under this Agreement;

2. sell, license, transfer, rent, loan, download, use, install, deploy, perform, modify, reproduce, distribute or disclose the Software (in whole or in part and whether done independently or as part of a compilation) for a Commercial Purpose;

3. post or make generally available the Software (in whole or in part) to individuals or a group of individuals who have not agreed to the terms and conditions of this Agreement;

4. share any license key or authentication information provided to a Licensee by Licensor with any third party to allow such party to access the Software; and

5. alter or remove any copyright notice or proprietary legend contained in or on the Software.

Paragraphs (a) though (e) of this Section 2.3 are collectively referred to as the "Restricted Activities").

2.4 Reproduction Obligations. Each Subscriber and Licensee agrees that any copy or distribution of the Software permitted under this Agreement shall contain the notices set forth in Exhibit A. In addition, to the extent a Licensee makes any copies of the Software or Documentation under this Agreement, each Subscriber and/or Licensee agrees to ensure that any and all such copies shall contain:

1. a copy of an appropriate copyright notice and all other applicable proprietary legends;

2. a disclaimer of any warranty consistent with this Agreement; and

3. any and all notices referencing this Agreement and absence of warranties.

CULA|tools

2.5 Distribution Obligations. The Distributable Files may be distributed pursuant to Section 2.1(b), and any use of such Distributable Files by a recipient must be governed by the terms and conditions of this Agreement. Each Subscriber must include a copy of this Agreement with every copy of the Distributable Files it distributes. Under no circumstance may a Subscriber or Licensee distribute CULA Premium or any files comprising CULA Basic not identified on Exhibit A. Each Subscriber must duplicate the notice in Exhibit A with the Distributable Files in a location (such as a relevant directory) where a user would be likely to look for such a notice.

3 Ownership of Software and Intellectual Property Rights. All right, title and interest to the Software and the Documentation, and all copies thereof, are and shall remain the exclusive property of Licensor and/or its licensors or suppliers. The Software is copyrighted and protected by the laws of the United States and other countries, and international treaty provisions. Licensor may make changes to the Software, or to items referenced therein, at any time and without notice, and Licensor is not obligated to support and/or update the Software or the Documentation unless otherwise agreed to herein. Except as otherwise expressly provided, Licensor grants no express or implied right or license (whether by implication, inducement, estoppel or otherwise) under any Licensor patents, copyrights, trademarks, or other intellectual property rights.

3.1 Feedback. While neither Party is required to provide the other party any suggestions, comments or other feedback regarding the Software, the Documentation or a Subscriber's and/or Licensee's use or implementation of the Software and/or Documentation ("Feedback"), to the extent a Subscriber or a Licensee provides Feedback to the Licensor, Licensor may use and include any Feedback so provided to improve the Software or other Licensor technologies and any new features, functionality, or performance based upon the Feedback that Licensor subsequently incorporates into its products shall be the sole and exclusive property of Licensor. Accordingly, if a Subscriber and/or Licensee provide Feedback, such Subscriber and/or Licensee hereby agrees that Licensor may freely use, reproduce, license, distribute, and otherwise commercialize the Feedback in the Software or other related technologies without the payment of any royalties or fees.

3.2 No Reverse Engineering and other Restrictions. In addition to agreeing to restrictions in Section 2.3 above, each Subscriber and/or Licensee shall not directly or indirectly: (i) sell, lease, redistribute or transfer any of the Software or Documentation; (ii) modify, translate, reverse engineer (except to the limited extent permitted by law), decompile, disassemble, create derivative works based on, sublicense, or distribute any of the Software; (iii) rent or lease any rights in any of the Software or Documentation in any form to any person; (iv) use any Software for the benefit of any third parties (e.g., in an ASP, outsourcing or service bureau relationship) or in any way other than in its intended manner; (v) remove, alter or obscure any proprietary or copyright notice, labels, or marks on the hardware components of the Software or Documentation; or (vi) disable or circumvent any access control or related security measure, process or procedure established with respect to the Software or any other part thereof. Each Subscriber and/or Licensee is responsible for all use of the Software and the Documentation and any downloading, installing and using the Software and for compliance with this Agreement; any breach by any user shall be deemed to have been made by the applicable Subscriber and/or Licensee.

4 Support. A Licensee may subscribe to Licensor's CULA Premium maintenance and support program by paying Licensor the then-applicable annual maintenance and support fee (the "Support Fee"). Upon payment of the Support Fee, Licensor shall provide Licensee with the applicable level of maintenance and support services set forth in the support program. Any CULA Premium updates provided to Licensee pursuant to the support program shall be deemed part of the CULA Premium and shall be licensed under the terms and conditions of the CULA Premium.

5 Payments. Licensee agrees to pay amounts invoiced by Licensor for any CULA Premium made available pursuant to this Agreement. If any authority imposes a duty, tax or similar levy (other than taxes based on Licensor's income), Licensee agrees to pay, or to promptly reimburse Licensor for, all such amounts. Unless otherwise indicated in an invoice, all Licensor invoices are payable thirty (30) days from the date of the invoice. Licensor reserves the right to charge a late payment in the event Licensee fails to remit payments when due.

6 Confidentiality. "Confidential Information" means any non-public technical or business information of a party, including without limitation any information relating to a party's techniques, algorithms, software, know-how, current and future products and services, research, engineering, vulnerabilities, designs, financial information, procurement requirements, manufacturing, customer lists, business forecasts, marketing plans and information. Each Party shall maintain in confidence all Confidential Information of the disclosing Party that is delivered to the receiving Party and will not use such Confidential Information except as expressly permitted herein. Each Party will take all reasonable measures to maintain the confidentiality of such Confidential Information, but in no event less than the measures it uses to protect its own Confidential Information.

6.1 Each Subscriber and/or Licensee hereby agrees that the Licensor shall be free to use any general knowledge, skills and experience, (including, but not limited to, ideas, concepts, know-how, or techniques) ("Residuals"), contained in any (i)_ Subscriber and/or Licensee Confidential Information, (ii) Feedback provided by a Subscriber and/or Licensee; (iii) Subscriber's and/or Licensee's products shared or disclosed to Licensor in connection with the Feedback, in each case, which are retained in the memories of Licensor's employees, agents, or contractors who have had access to such materials. Licensor shall have no obligation to limit or restrict the assignment of its employees, agents or contractors or to pay royalties for any work resulting from the use of Residuals.

7 Limited Warranty and Disclaimer.

7.1 NO WARRANTIES. THE SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND, INCLUDING WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. Licensor does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the Software. Licensor does not represent that errors or other defects will be identified or corrected.

7.2 LIMITATION OF LIABILITY. EXCEPT WITH RESPECT TO THE MISUSE OF THE OTHER PARTY'S INTELLECTUAL PROPERTY OR DISCLOSURE OF THE OTHER PARTY'S CONFIDENTIAL INFORMATION IN BREACH OF THIS AGREEMENT, IN NO EVENT SHALL LICENSOR, SUBSIDIARIES, LICENSORS, OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, INDIRECT, LOST PROFITS, CONSEQUENTIAL, BUSINESS INTERRUPTION OR LOST INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS PROHIBIT EXCLUSION OR LIMITATION OF LIABILITY FOR IMPLIED WARRANTIES OR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO SCRIBER AND/OR LICENSEE. SUBSCRIBER AND/OR LICENSEE MAY ALSO HAVE OTHER LEGAL RIGHTS THAT VARY FROM JURISDICTION TO JURISDICTION. NOTWITHSTANDING THE FOREGOING, LICENSOR'S AGGREGATE LIABILITY ARISING OUT OF THIS AGREEMENT SHALL NOT EXCEED ONE HUNDRED UNITED STATES DOLLARS (USD$100).

8 Term; Termination.

8.1 Term. The term of this Agreement shall commence on the Effective Date and shall expire on the first (1st) anniversary of the Effective Date (the "Initial Term"). This Agreement shall automatically renew for successive one (1) year periods (the "Renewal Term," and together with the Initial Term, the "Term") unless Licensee provides Licensor with notice of non-renewal at least thirty (30) calendar days prior to its expiration.

8.2 Termination.

1. Breach. Either Party may terminate this Agreement upon thirty (30) days' prior written notice if the other Party materially breaches this Agreement and does not cure such breach within thirty (30) days following receipt of notice specifying the breach.

2. Insolvency. Either Party may also have the right to terminate this Agreement in the event the other

party (i) becomes insolvent, (ii) becomes subject to a petition in bankruptcy filed by or against it that is not dismissed within thirty days of the filing of such petition, (iii) is placed under the control of a receiver, liquidator or committee of creditors, or (iv) dissolves, ceases to function as a going concern or to conduct its business in the normal course.

8.3 Effect of Termination. Upon the expiration or termination of this Agreement, Customer agrees to pay all amounts accrued or otherwise owing to Licensor on the date of termination, and each Party shall return, or certify the destruction of, the Confidential Information of the other Party. In the event a support program is in place at the effective date of termination, Licensor agrees to continue providing maintenance and support under the terms of this Agreement and the applicable Support Plan through the expiration date of such Support Plan. Termination in accordance with this Agreement shall be without prejudice to any other rights or remedies of the Parties.

9 Miscellaneous.

9.1 Legal Compliance; Restricted Rights. Each Party agrees to comply with all applicable Laws. Without limiting the foregoing, Customer agrees to comply with all U.S. export Laws and applicable export Laws of its locality (if Customer is not located in the United States), and Customer agrees not to export any Network Appliances, Software or other materials provided by Licensor without first obtaining all required authorizations or licenses. The Network Appliances and Software provided to the United States government are provided with only "LIMITED RIGHTS" and "RESTRICTED RIGHTS" as defined in FAR 52.227-14 if the commercial terms are deemed not to apply.

9.2 Governing Law; Severability. This Agreement shall be governed by the laws of the State of New Jersey, USA, without regard to choice-of-law provisions. If any provision of this Agreement is held to be illegal or unenforceable for any reason, then such provision shall be deemed to be restated so as to be enforceable to the maximum extent permissible under law, and the remainder of this Agreement shall remain in full force and effect. Each Subscriber and/Licensee and Licensor agree that this Agreement shall not be governed by the U.N. Convention on Contracts for the International Sale of Goods. Any and all proceedings relating to the subject matter of this Agreement shall be maintained in the courts of the State of Delaware or Federal District Courts sitting in the District of Delaware, which courts shall have exclusive jurisdiction for such purpose, and Subscriber and/or Licensee hereby consents to the personal jurisdiction of such courts.

9.3 Notices. Any notices under this Agreement will be personally delivered or sent by certified or registered mail, return receipt requested, or by nationally recognized overnight express courier, to the address specified herein or such other address as a Party may specify in writing. Such notices will be effective upon receipt, which may be shown by confirmation of delivery. All notices to Licensor shall be sent to the attention of General Counsel (unless otherwise specified by Licensor).

9.4 Assignment. Neither Party may assign or otherwise transfer this Agreement without the other Party's prior written consent, which consent shall not be unreasonably withheld, conditioned or delayed. Notwithstanding the foregoing, either Party may assign this Agreement without the consent of the other Party if a majority of its outstanding voting capital stock is sold to a third party, or if it sells all or substantially all of its assets or if there is otherwise a change of control. This Agreement shall be binding upon and inure to the benefit of the Parties' successors and permitted assigns.

9.5 Force Majeure. Neither Party shall be liable for any delay or failure due to a force majeure event and other causes beyond its reasonable control. This provision shall not apply to any of Customer's payment obligations.

9.6 Counterparts. This Agreement may be executed in counterparts, each of which will be deemed an original, but all of which together will constitute one and the same instrument. This Agreement may be executed by facsimile or scanned signatures.

9.7 General. This Agreement, including its exhibits (all of which are incorporated herein), are collectively the Parties' complete agreement regarding its subject matter, superseding any prior oral or written communications. Amendments or changes to this Agreement must be in mutually executed writings to be

effective. The individual executing this Agreement on behalf of the Customer has the requisite power and authority to sign this Agreement on behalf of Customer. The Parties agree that, to the extent any Customer purchase or sales order contains terms or conditions that conflict with, or supplement, this Agreement, such terms and conditions shall be void and have no effect, and the provisions of this Agreement shall control. Unless otherwise expressly set forth in an exhibit that is executed by the Parties, this Agreement shall control in the event of any conflict with an exhibit. Sections 2, 3, 5, 6, 7, 8 and 9 and all warranty disclaimers, use restrictions and provisions relating to Licensor's intellectual property ownership, shall survive the termination or expiration of this Agreement. The Parties are independent contractors for all purposes under this Agreement.

EXHIBIT A

CULA Basic and CULA Premium Software License Agreement

The contents of this file are subject to the Software License Agreement (the "Agreement"). You may not use this file except in compliance with the Agreement. You may obtain a copy of the Agreement at http://www.culatools.com/licenses/license.html.

Software distributed under the Agreement is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the Agreement for the specific language governing rights and limitations under the Agreement.

The developer of the CULA Basic and CULA Premium is EM Photonics, Inc., a Delaware corporation.

2008 - 2009 Copyright EM Photonics, Inc. All Rights Reserved.

CULATOOLS(TM), CULA(TM), the EM Photonics logo, and certain other trademarks and logos are trademarks or registered trademarks of EM Photonics, Inc. in the United States and other countries.

Note: A printer friendly version of this Agreement is available in RTF format.

CULA Basic Files Re-distributable Pursuant to Section 2.1 (b).

   cula.dll

   cula_fortran.dll

   cula.so.*

   cula_fortran.so.*

   cula.dylib

   cula_fortran.dylib

GETTING STARTED

3.1 Obtaining CULA

CULA can be downloaded from www.culatools.com. The Basic release is available to those who register on the website. CULA premium, which add dozens of extra LAPACK routines as well as double-precision and double-precision complex data formats, can be purchased from the CULAtools website at www.culatools.com/purchase.

For vendors interested in CULA Commercial, please contact EM Photonics directly www.culatools.com/contact.

3.2 System Requirements

CULA utilizes *CUDA* on an NVIDIA GPU to perform linear algebra operations. Therefore, an NVIDIA GPU with *CUDA* support is required to use CULA. A list of supported GPUs can be found on NVIDIA's CUDA Enabled webpage.

Support for double-precision operations requires a GPU that supports *CUDA* Compute Model 1.3. To find out what Compute Model your GPU supports, please refer to the NVIDIA CUDA Programming Guide.

Note: CULA's performance is primarily influenced by the processing power of your system's GPU, and as such a more powerful graphics card will yield better performance.

3.3 Installation

Installation is completed via the downloadable installation packages. To install CULA, refer to the section below that applies to your system.

Windows

Run the CULA installer and when prompted select the location to which to install. The default install location is *c:\Program Files\CULA*. The CULA uninstaller, accessible via the Start Menu, will remove CULA from your system.

Linux

It is recommended that you run the CULA installer as an administrator in order to install to a system-level directory. The default install location is */usr/local/cula*.

> **Mac OS X Leopard**
>
> Open the CULA .dmg file and run the installer located inside. The default install location is */usr/local/cula*. There is no uninstaller on OS X, but you can remove the folder to which you installed CULA.

Note: You may wish to set up environment variables to common CULA locations. More details are available in the *Configuring Your Environment* chapter.

3.4 Compiling with CULA

As described in the *Introduction*, CULA presents four interfaces. This section describes the differences between the Standard and Device interfaces. The third interface (Fortran) is discussed in a later section.

CULA presents two main *C* headers, *culapack.h* for the standard interface and *culapackdevice.h* for the device interface. These files differ in the type of memory they accept. For the Standard interface, the pointers that these functions accept are to main memory, while in Device interface the pointers are to GPU memory. A convenience header *cula.h* is provided and includes all C functionality.

CULA's Standard interface automatically transfers the contents of the main memory to the GPU before beginning processing, and transfers the results back to the main memory upon completion of the operation. Due to internal memory mechanisms, this can be many times more efficient than allowing the user to pre-load the data to their GPU. However, if the programmer is building a closed-loop GPU system, the data may already be on-card and the CULA Device interface may be more convenient.

CULA's Device interface operates on pre-allocated GPU memory. This interface is intended for those who have pre-existing systems or are trying to build a closed-loop GPU processing system. These inputs are located in GPU memory and the results are placed in GPU memory. Allocation and transfers of data are the responsibility of the programmer and are performed using the *CUDA* API.

3.5 Linking to CULA

CULA provides a link-time stub library, but is otherwise built as a shared library. Applications should link against the following libraries:

> **Windows**
>
> Choose to link against cula.lib as a link-time option.

> **Linux / Mac OS X Leopard**
>
> Add `-lcula` to your program's link line.

CULA is built as a shared library, and as such it must be visible to your runtime system. This requires that the shared library is located in a directory that is a member of your system's runtime library path . For more detailed information regarding operating-system-specific linking procedures, please refer to the *Configuring Your Environment* chapter.

CULA's example projects are a good resource for learning how to set up CULA for your own project.

Note: CULA is built against NVIDIA CUDA 2.3 and ships with a copy of the CUDA 2.3 redistributable files. If you have a different version of CUDA installed, you **must** ensure that the CUDA runtime libraries shipped with CULA are the first visible copies to your CULA program. This can be accmplished by placing the CULA *bin* path earlier in your system PATH than any CUDA *bin* path. If a non-CUDA-2.3 runtime loads first, you will experience CULA errors.

3.6 Uninstallation

After installation, CULA leaves a record to uninstall itself easily. To uninstall CULA, refer to the section below that applies to your system.

Windows

From the Start Menu, navigate to the *CULA* menu entry under *Programs*, and select the Uninstall option. The CULA uninstaller will remove CULA from your system.

Linux

Run the CULA installer, providing an 'uninstall' argument.
e.g. ./cula.run uninstall

Mac OS X Leopard

There is no uninstallation on OS X, but you can remove the folder to which you installed CULA for a complete uninstall.

Note: If you have created environment variables with references to CULA, you may wish to remove them after uninstallation.

DIFFERENCES BETWEEN CULA AND LAPACK

Unlike LAPACK, CULA uses both your system's CPU and GPU resources. Because of this difference, CULA diverges slightly from the original LAPACK interface. CULA reduces some of the complexities of traditional LAPACK interfaces, while providing the flexibility you need to manage the GPU as a compute device. This section describes the differences between CULA and LAPACK and how they affect your program.

For a discussion on individual routines, see the *CULA Reference Manual*.

4.1 Naming Conventions

CULA follows a function naming conventions that is similar but different to that of LAPACK. The primary reason for this different is so that the accelerated routines in CULA will be able to link cleanly against other linear algebra packages that supply routines that are not present in CULA. This restriction comes from linkage rules in C/C++ that dictate that there can only be one function of a given name within a program. If CULA used names that are equivalent to those of other packages, this would prevent users from compiling CULA and another linear algebra package in the same program.

CULA's routines are named using a system where:

- The function begins with `cula` (in lowercase). Device interface functions use an extended prefix of `culaDevice`.

- The next character (in uppercase) represents the data type.

- The following two characters (in lowercase) represent the matrix type.

- The final two to three characters (in lowercase) represent the computation performed by the routine.

For example, the following call will perform CULA's single-precision QR factorization on a general matrix.

The following table outlines the matrix types specific to CULA.

Abbreviation	Matrix Type
bd	Bidiagonal
ge	General
gg	General matrices, generalized problem
or	(Real) Orthogonal
tr	Triangular
un	(Complex) Unitary

The following table lists the suffix for some common routines CULA provides.

Abbreviation	Computation Routine
trf	Compute a triangular factorization
sv	Factor the matrix and solve a system of equations
qrf	Compute a QR factorization without pivoting
svd	Compute the singular value decomposition
ls	Solve over- or under-determined linear system

The full list of abbreviations can be found at the LAPACK website.

4.2 Calling Conventions

In addition to using different function names, CULA has moved away from the LAPACK interface and takes arguments by value rather than by reference, which is a convention more familiar to C/C++ programmers. Additionally, this simplifies calling code by reducing the need for some temporary variables.

For example, compare a call to a CULA function with one that uses traditional LAPACK conventions:

```
// Common variables
...

// CULA
culaStatus s;
s = culaSgesvd('O', 'N', m, m, a, lda, s, u, ldu, vt, ldvt);

// Traditional
char jobu = 'O';
char jobvt = 'N';
int info;
sgesvd(&jobu, &jobvt, &m, &n, a, &lda, s, u, &ldu, vt, &ldvt, &info);
```

Compared the to the traditional code, CULA uses fewer lines, has a shorter call line, and is no less clear than the traditional usage – one might argue that it is even more clear because there is less code to inspect.

In addition to the Standard and Device interfaces, CULA has a third interface, **Fortran**, is intended to be used in Fortran programs. This interface follows the conventions of Fortran programming by taking all parameters by reference rather than by value. For more information on this interface, see the 'fortran_interface.txt' document in the 'doc' directory.

With version 1.1, CULA introduced the **Bridge** interface. This interface is targeted at users who are porting existing linear algebra codes to a GPU-accelerated environment. The **Bridge** interface provides a migration path by matching the function names and signatures of several popular linear algebra packages. It additionally provides a fallback to CPU execution when a user does not have a GPU or when problem size is too small to take advantage of GPU execution. For more information on this interface, see the 'bridge_interface.txt' document in the 'doc' directory.

4.3 Data Type Support

Different versions of CULA support different data types. CULA Basic supports both single-precision real and complex. CULA Premium and Commercial support all LAPACK data types.

CULA provides support for float and float-complex data types by using C's built-in `float` and the CULA type `culaFloatComplex`. Similarly, CULA provides support for double and double-complex data types by using C's built-in `double` and the CULA type `culaDoubleComplex`, though these data types require support from the GPU (see *System Requirements*).

Prefix	Data Type	CULA Standard Interface Type	CULA Device Interface Type
S	Single Precision Real	`culaFloat`	`culaDeviceFloat`
C	Single Precision Complex	`culaFloatComplex`	`culaDeviceFloatComplex`
D	Double Precision Real	`culaDouble`	`culaDeviceDouble`
Z	Double Precision Complex	`culaDoubleComplex`	`culaDeviceDoubleComplex`

Note: *CUDA* complex types may be used in place of CULA types by defining the preprocessor macro CULA_USE_CUDA_COMPLEX before including a CULA header.

4.4 Error Handling

LAPACK's error handling is accomplished through an *info* parameter. Upon return, this parameter specifies whether or not the function succeeded (return value of *0*) and if it did not, indicates the particular parameter that is in error by stating the parameter number. Additionally, LAPACK may print an error to *stdout*.

CULA uses a system similar to LAPACK's error handling but adds additional functionality. Instead of requiring an *info* parameter, CULA returns the success or failure of a function using a *culaStatus* enumeration. This return code specifies conditions that the *info* parameter does not cover, including:

- a valid co-processing device is not available;
- insufficient memory is available to continue;
- a requested feature is not available;
- the co-processing device is missing a required feature to complete the requested operation.

Two classes of errors that are typically reported by LAPACK are those in arguments passed to a function and those resulting from malformed data. CULA reports these errors using *culaArgumentError* and *culaDataError* return values. When this value is returned, further information about this error can be requested by calling *culaGetErrorInfo*, which returns an integer equivalent to that which would be returned by the analogous LAPACK function. Unlike LAPACK, CULA will not print information to *stdout*.

4.5 Workspaces

Many LAPACK functions require a workspace for internal operation. For those LAPACK functions that utilize a workspace, workspace sizes are queried by providing a -1 argument to what is typically an *LWORK* parameter. Upon inspecting this parameter, the LAPACK function will determine the workspace required for this particular problem size and will return the value in the *WORK* parameter. LAPACK (and other similar packages) then require the programmer to provide a pointer to memory of sufficient size, which often requires that the programmer allocate new memory.

CULA uses both main and GPU workspace memories, and as such, LAPACK's workspace query is not appropriate, as the LAPACK interface allows for the specification of only one workspace. Instead of providing a more complicated interface that adds parameters for both main and GPU workspace memories, CULA requires neither. Instead, any

workspaces that are required are allocated and tracked internally. This organization yields no significant performance loss, and furthermore reduces the number of function calls by removing the need for a workspace query.

Note: Any workspaces that have been allocated internally may be cleared by calling `culaFreeBuffers()`.

PROGRAMMING CONSIDERATIONS

This chapter contains a variety of subjects related to programming using CULA.

5.1 Matrix Storage

When providing data to CULA routines, it is important to consider the manner in which the data is stored in memory. In particular, there are two important requirements to ensure correct results:

- Data must be in "column-major" order.
- Complex data must be interleaved.

The following sections will briefly explain these requirements.

5.1.1 Column-Major Ordering

CULA routines expect that any data provided will be stored in column-major order. While this storage scheme is also expected for LAPACK routines, column-major order may be uncommon to many programmers. Consider the following *M-by-N* matrix.
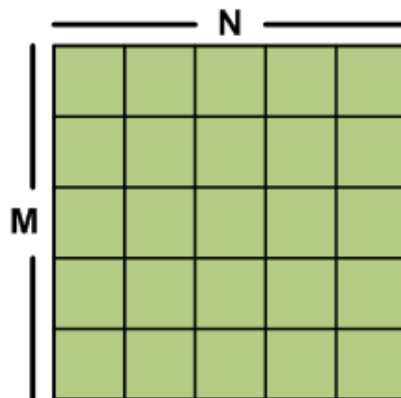

Figure 5.1: A basic *M x N* matrix.

When storing the matrix in memory, two methods are frequently used: row-major and column-major.

For many C programmers, a familiar storage pattern would involve storing the elements of a particular row first, followed by the elements of successive rows. In this storage scheme, elements of a row are contiguous in memory, while elements of a column are not. C, for example, is row-major storage by default.
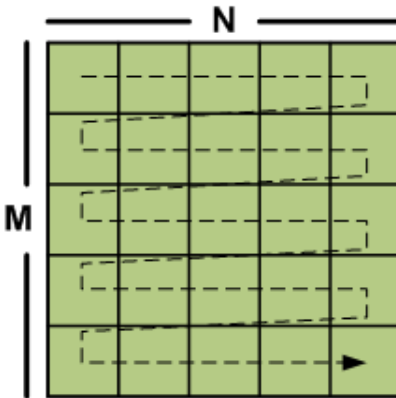
Figure 5.2: A row-major ordered matrix. Elements are stored in memory in the order shown by the arrow.

Column-major ordering is the opposite of this because elements of the matrix are now stored by column, rather than by row. In this storage scheme, elements of a column are contiguous in memory, while elements of a row are not.
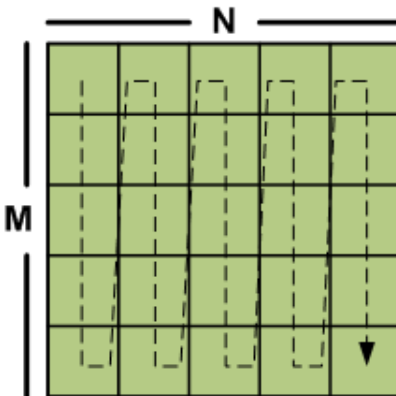

Figure 5.3: A column-major ordered matrix. Elements are stored in memory in the order shown by the arrow.

As mentioned previously, CULA expects data to be represented in column-major order. Performing a transpose on the row-major data will convert it to column-major and vice-versa.

Note: Many CULA functions take a "leading dimension" argument after matrix arguments. For column-major data, the leading dimensionthe leading dimension is equal to the length of a column, or equivalently, the number of rows in the matrix. This is the length of the allocated memory and may be larger than the memory involved in a particular computation. For submatrix operations or pitched allocations, remember to report the leading dimension parameter as the number of *allocated* rows in the matrix.

Note: Note the differences in addressing row-major and column-major order.

Given an element of matrix A (with leading dimension LDA) located at row "i" and column "j," this element would be accessed as follows:

- `A[j*LDA + i]` for row-major data.
- `A[i*LDA + j]` for column-major data in CULA.

5.1.2 Complex Data Types

When working with complex data, CULA expects real and complex portions of the matrix to be interleaved. In other words, for any particular element, the real and complex parts are adjacent in memory, with the complex part being stored after the real part.
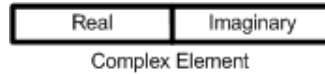

Figure 5.4: This figure shows the packing of a complex data element. In memory, the real part of an element is directly followed by the complex component of that element.

5.2 Optimizing for Performance

CULA is specifically designed to leverage the massively parallel computational resources of the GPU, with a particular focus on large problems whose execution on a standard CPU is too time consuming. As such, several considerations should be taken into account when applying CULA routines to maximize overall performance gains.

5.2.1 Problem Size

Prior to the application of CULA routines it is important to consider problem size. As a general rule, applying CULA for larger problems will maximize performance gains with respect to other computational linear algebra packages. For example, consider the *SGEQRF Speedups* Figure.
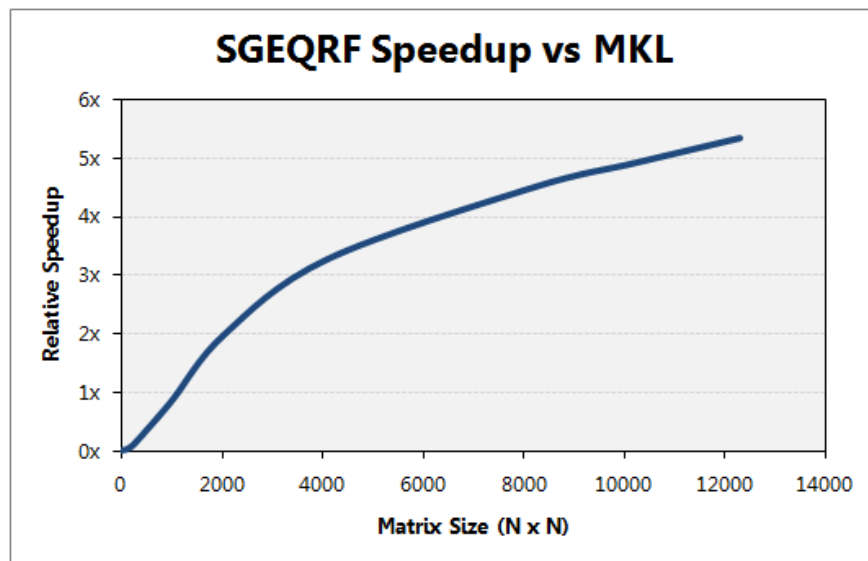
Figure 5.5: This figure shows the speedup for a selected routine versus problem size. High speedups are attained for larger problem sizes.

The speedup chart illustrates the performance of CULA's QR decomposition with respect to Intel's Math Kernel Library (MKL). Note that for the smallest problem sizes, no performance gains are seen; however, problem complexity quickly scales in such a manner that the GPU is able to outperform the CPU, for even modest problem sizes.

Note: The maximum problem size is constrained by the data type in use and the maximum GPU memory. For example, the maximum size for a problem that uses double-precision complex data is roughly one fourth of the maximum problem size of a single-precision problem for the same matrix dimensions, since the size of these data types differ by a factor of four.

5.2.2 Accuracy Requirements

CULA offers both single and double-precision floating point support for its included routines. While the latest NVIDIA GPU hardware offers support for both of these data types, it should be noted that current NVIDIA GPU hardware performs best when operating on single-precision data.

When applying CULA routines, the accuracy requirements of the program should be evaluated. If extended precision is required, then the use of double-precision routines is advised. For programs with more relaxed accuracy requirements, however, additional performance can be achieved at the cost of accuracy through the use of single-precision routines.

5.3 Using the Device Interface

CULA's device interface allows a user to directly control GPU memory. This is useful for closed-loop GPU systems that do large sections of processing on the GPU. The device interface is not recommended to those without *CUDA* experience.

When using the device interface, include *culapackdevice.h* instead of *culapack.h*; see the *Matrix Storage* section for differences in naming conventions between the Standard and Device interfaces. Memory allocation is handled via cudaMalloc and cudaFree, available in the *CUDA* toolkit. If the programmer uses pitched memory, it is up to that programmer to ensure that their allocations are appropriate.

Note: CULA's Premium version provides specialized allocation functions that pitch data to the optimal size for CULA. These functions are `culaDeviceMalloc()` and `culaDeviceFree()`, found in the *culadevice.h* header of CULA Premium.

5.4 Thread Safety/Multi-GPU Operation

Currently, the default operating mode of CULA is to operate on only one GPU. Aside from this default mode, a programmer can manually control the way that CULA is used across the various GPU devices in their system. For a given function call, CULA currently supports single-GPU operation only. However, function calls may be made from several threads because all API functions are designed to be thread safe. Thus, you may write a program that is multithreaded, with various threads making accesses to CULA to perform their individual linear algebra operations.

By default, CULA binds to the best GPU in a system. However, this binding may be manually controlled by using `culaSelectDevice` (found in culadevice.h). It is important to note that if a given thread is already bound to a particular GPU device, CULA cannot reset this binding, as this is a restriction of *CUDA*.

To use multiple GPUs, multiple threads must be created. Each thread must be bound to a different GPU device by calling `culaSelectDevice`, with a different GPU id for each thread. After doing so, call `culaInitialize` in each thread, and proceed to call CULA functions as you need. Before each thread exits, ensure that `culaShutdown` is called in each.

When using CULA, normal parallel programming rules apply. The section below outlines a few of these considerations for each of CULA's interfaces.

Standard / Fortran Interface

The Standard interface uses (traditional) main memory pointers. It is up to the programmer to ensure that CULA does not concurrently write to overlapping sections of memory; doing so will yield undefined results.

> **Device Interface**
>
> The *CUDA* programming model specifies that a given thread may only communicate with a single GPU in a system. To follow this requirement, any call to a *CUDA* API function or a CULA function will "bind" that thread to that GPU (except where specified). These calls include memory allocations functions. A device pointer is only valid on a particular GPU, so care must be taken when calling CULA in a multithreaded environment. The important point is to only pass device pointers that were allocated in that particular thread context. To learn which device a thread is bound to, you may call `culaGetExecutingDevice()` routine found in culadevice.h.

5.5 Developing in C++

CULA provides a set of generic functions for use in C++ applications. These functions are overloaded to select a function according to the data type of its operand, allowing generic code. For every function in the *culapack.h* and *culapackdevice.h* headers, there is a generic version of that function in the *culapack.hpp* and *culapackdevice.hpp* headers. The generic functions will determine the type of the calling data and dispatch the operation to the appropriate typed function. Similarly to the C interfaces, a *cula.hpp* file is provided as a convenience header to include both the Standard and Device interfaces simultaneously if desired.

Please see the *Examples* chapter for an example of this interface.

EXAMPLES

The CULA installation contains several examples that show how to use CULA. These examples are contained in the 'examples' directory.

This section lists several of the common use cases for CULA functions.

6.1 Initialization and Shutdown

```c
#include <cula.h>

culaStatus s;

s = culaInitialize();
if(s != culaNoError)
{
    printf("%s\n", culaGetErrorString(s));
    /* ... Error Handling ... */
}

/* ... Your code ... */

culaShutdown();
```

6.2 Argument Errors

```c
#include <cula.h>

culaStatus s;

s = culaSgeqrf(-1, -1, NULL, -1, NULL); /* obviously wrong */
if(s != culaNoError)
{
    if(s == culaArgumentError)
        printf("Argument %d has an illegal value\n", culaGetErrorInfo());
    else
        printf("%s\n", culaGetErrorString(s));
}
```

6.3 Data Errors

```
#include <cula.h>

float *A = malloc(20*20*sizeof(float));
memset(A, 0, 20*20*sizeof(float)); /* singular matrix, illegal for LU (getrf) */
int ipiv[20];
s = culaSgetrf(20, 20, A, 20, ipiv);
if( s != culaNoError )
{
    if( s == culaDataError )
        printf("Data error with code %d, please see LAPACK documentation\n",
            culaGetErrorInfo());
    else
        printf("%s\n", culaGetErrorString(s));
}
```

6.4 Using the C++ Interface

```
#include <cula.hpp>

template<class T>
void GenericLU(T *A, int N) // type of T will be determined by the compiler
{
    std::vector<int> piv(N);

    // no need for type specifier – determined automatically on overloads
    culaStatus s = culaGetrf(N, N, A, N, &ipiv[0]);

    // check errors
}
```

CONFIGURING YOUR ENVIRONMENT

This section describes how to set up CULA using common tools, such as Microsoft® Visual Studio®, as well as command line tools for Linux and Mac OS X.

7.1 Microsoft Visual Studio

This section describes how to configure Microsoft Visual Studio to use CULA. Before following the steps within this section, take note of where you installed CULA (the default is *C:\Program Files\CULA*). To set up Visual Studio, you will need to set both Global- and Project-level settings. Each of these steps is described in the sections below.

7.1.1 Global Settings

When inside Visual Studio, navigate to the menu bar and select *Tools > Options*. A window will open that offers several options; in this window, navigate to *Projects and Solutions > VC++ Directories*. From this dialog you will be able to configure global executable, include, and library paths, which will allow any project that you create to use CULA.

The table below specifies the recommended settings for the various directories that the *VC++ Directories* dialog makes available. When setting up your environment, prepend the path of your CULA installation to each of the entries in the table below. For example, to set the include path for a typical installation, enter *C:\Program Files\CULA\include* for the *Include Files* field.

Option	Win32	x64
Executable Files	bin	bin64
Include Files	include	include
Library Files	lib	lib64

With these global settings complete, Visual Studio will be able to include CULA files in your application. Before you can compile and link an application that uses CULA, however, you will need to set up your project to link CULA.

7.1.2 Project Settings

To use CULA, you must instruct Visual Studio to link CULA to your application. To do this, right-click on your project and select *Properties*. From here, navigate to *Configuration Properties > Linker > Input*. In the *Additional Dependencies* field, enter "cula.lib".

On the Windows platform, CULA's libraries are distributed as a dynamic link library (DLL) (*cula.dll*) and an import library (*cula.lib*), located in the *bin* and *lib* directories of the CULA installation, respectively. By Linking *cula.lib*, you are instructing Visual Studio to make an association between your application and the CULA DLL, which will allow your application to use the code that is contained within the CULA DLL.

7.1.3 Runtime Path

CULA is built as a dynamically linked library, and as such it must be visible to your runtime system. This requires that cula.dll and its supporting dll's are located in a directory that is a member of your system's runtime path. On Windows, you may do one of several things:

1. Add CULA_BIN_PATH_32 or CULA_BIN_PATH_64 to your PATH environment variable.

2. Copy cula.dll and its dll's to the working directory or your project's executable.

7.2 Linux / Mac OS X - Command Line

On a Linux system, a common way of building software is by using command line tools. This section describes how a project that is command line driven can be configured to use CULA.

7.2.1 Configure Environment Variables

The first step in this process is to set up environment variables so that your build scripts can infer the location of CULA.

On a Linux or Mac OS X system, a simple way to set up CULA to use environment variables. For example, on a system that uses the *bourne* (*sh*) or *bash* shells, add the following lines to an appropriate shell configuration file (e.g. *.bashrc*).

```
export CULA_ROOT=/usr/local/cula
export CULA_INC_PATH=$CULA_ROOT/include
export CULA_BIN_PATH_32=$CULA_ROOT/bin
export CULA_BIN_PATH_64=$CULA_ROOT/bin64
export CULA_LIB_PATH_32=$CULA_ROOT/lib
export CULA_LIB_PATH_64=$CULA_ROOT/lib64
```

(where `CULA_ROOT` is customized to the location you chose to install CULA)

After setting environment variables, you can now configure your build scripts to use CULA.

Note: You may need to reload your shell before you can use these variables.

7.2.2 Configure Project Paths

This section describes how to set up the *gcc* compiler to include CULA in your application. When compiling an application, you will typically need to add the following arguments to your compiler's argument list:

Item	Command
Include Path	`-I$CULA_INC_PATH`
Library Path (32-bit arch)	`-L$CULA_LIB_PATH_32`
Library Path (64-bit arch)	`-L$CULA_LIB_PATH_64`
Libraries to Link against	`-lcula`

For a 32-bit compile:

```
gcc ... -I$CULA_INC_PATH -L$CULA_LIB_PATH_32 -lcula -lcublas -lcudart ...
```

For a 64-bit compile (not applicable to Mac OS X):

```
gcc ... -I$CULA_INC_PATH -L$CULA_LIB_PATH_64 -lcula -lcublas -lcudart ...
```

7.2.3 Runtime Path

CULA is built as a shared library, and as such it must be visible to your runtime system. This requires that CULA's shared libraries are located in a directory that is a member of your system's runtime library path. On Linux, you may do one of several things:

1. Add CULA_LIB_PATH_32 or CULA_LIB_PATH_64 to your LD_LIBRARY_PATH environment variable.

2. Edit your system's ld.so.conf (found in /etc) to include either CULA_LIB_PATH_32 or CULA_LIB_PATH_64.

On the Mac OS X platform, you must edit the DYLD_LIBRARY_PATH environment variable for your shell, as above.

TROUBLESHOOTING

This section lists solutions for common problems encountered when using CULA and describes your support options.

8.1 Common Issues

How do I report information about my system?

Run one of the *sysinfo.bat* or *sysinfo.sh* scripts, for Windows or Linux/Mac OS X systems, respectively. The information these scripts report can be uploaded as an attachment to your support requests or forum posts and will aid us greatly in diagnosing your problems.

cuda.dll or libcula.so could not be found

Your runtime system has not been set up to locate CULA. See the *Linking to CULA* section.

CULA returns culaArgumentError.

Your input to one of CULA's routines is incorrect. The faulty parameter is reported by culaGetErrorInfo; ensure that all of your parameters are correct. See the *Error Handling* section.

CULA returns culaDataError.

An input to one of CULA's routines is malformed or singular. More information about the error is reported by culaGetErrorInfo. See the *Error Handling* section.

CULA returns culaRuntimeError.

The most common cause of this error is related to improperly configured or old GPU drivers. Ensure that you system is up to date by installing the latest drivers from NVIDIA.

If the problem persists, please file a bug report.

An LAPACK routine I need is missing. What should I do?

LAPACK is a very large library and we have only written a subset of its capabilities so far. CULA Premium and CULA Commercial are constantly growing as new functions are added. If there is function you would like to see added, contact us on our forums and voice your opinion on which functions you use.

I'm having problems with my GPU and/or video drivers. Can you help?

Problems specific to GPU devices and their drivers should be handled by NVIDIA. The latest drivers can always be obtained directly from NVIDIA on their Download page. CULA requires CUDA 2.3 compatible drivers (or newer) to be installed on your system. On Linux, this requires driver version 190.18 or newer, and for Windows it is version 190.38 or newer. On Mac OS X, please use "cudadriver_2.3.0_macos.pkg" or newer.

I think I found a bug. What should I do?

> First, be sure that you are experiencing a software bug and not an issue related to faulty hardware or drivers. If you are still reasonably sure you have encountered a bug,

8.2 Support Options

If none of the entries above solve your issue, you can seek technical support. See your license level below to learn about your support options.

- **Basic** - Basic users may post problems to the CULA forums for community support.

- **Premium** - Premium users can submit support tickets through the CULA website to receive personalized help.

- **Commercial** - Commerical users have direct support. Please contact your support representative directly for help with your issue.

When reporting a problem, make sure to include the following information:

- System Information (see *Common Issues*)

- CULA Version

- Version of NVIDIA® *CUDA* Toolkit installed, if any

- Problem Description (with code if applicable)

CHANGELOG

9.1 Release 1.1 Final (November 25, 2009)

All Versions

- Improved: Removed culaInitialize() delay

- Improved: GEQRF performance up to 20% improvement for users with older CPUs

- Fixed: Correction to Fortran example Makefile to specify "arch" parameter

Basic

- Improved: SVD significant memory usage reduction

- Improved: GELS stability for N > M case

Premium

- Improved: GELQF performance increase 2-3x

- Improved: GEHRD/GERQF/ORGLQ/ORGQR performance increased by 10-20%

- Improved: GEHRD routine accurate for size N==1

9.2 Release 1.1 Beta (November 13, 2009)

All Versions

- Feature: Mac OS X 10.5 Leopard "preview" release - single precision only

- Feature: New "Bridge" interface provides for easy and seamless porting of existing LAPACK/MKL/ACML applications (see doc/bridge_interface.txt)

- Feature: New document describing full CULA API

- Feature: New function culaSelectDevice to set executing device

- Feature: New "gesv" example shows operation of all S/C/D/Z data types

- Feature: New "multigpu" example showing multi-GPU CULA operation

- Feature: New "bridge" example showing usage of the Bridge interface

- Improved: SVD optimized for non-square cases

- Improved: Documentation clarified on error conditions and codes

- Improved: Stronger error reporting from example projects

- Improved: culaInitialize detects and reports if driver/runtime version are inadequate

- Improved: Documentation clearer on thread safety issues

- Fixed: CULA can now handle extremely non-square matrices (eg 500000x16)

- Fixed: An error in the "benchmark" example causing it to ignore user arguments

- Fixed: Properly reporting cudaErrorMemoryValueTooLarge as culaInsufficientMemory

Basic

- Improved: GESV performance increased by up to 30%

- Improved: Stability of GELS in certain cases

- Improved: Stability of SVD in certain cases

Premium

- Feature: Implemented geev (general Eigensolver) in S/D/C/Z precisions

- Feature: Implemented gehrd (general Hessenberg reduction) in S/D/C/Z precisions

- Feature: Implemented orghr

- Feature: .hpp headers have name overloads of ORG/UNG functions

- Fixed: Host interface "ORG" functions different results from device interface

9.3 Release 1.0 Final (September 30, 2009)

Basic

- Feature: All functions feature complex variants

- Fixed: Crash related to getrs pivot array

Premium

- Feature: All functions implemented in all supported data types

9.4 Release 1.0 Beta 3 (September 15, 2009)

All Versions

- Feature: New documentation section on specific routine conventions

- Improved: Updated sysinfo script with more descriptive output

- Improved: Added example that demonstrates the device interface

- Fixed: Various corrections for small-matrix inputs, especially M=N=1

- Fixed: culaInitialize now sets environment variable KMP_DUPLICATE_LIB_OK

Basic

- Feature: Complex geqrf included

- Feature: Added culaGetDeviceCount to report the number of available devices

- Feature: Added culaGetDeviceInfo to report information about a device

- Feature: Added culaGetExecutingDevice to report the executing device

- Fixed: Further corrections for unitary output in gesvd for all job codes

Premium

- Feature: New functions culaDeviceMalloc/culaDeviceFree in culadevice.h

- Fixed: Orglq and orgqr should behave more reliably

9.5 Release 1.0 Beta 2 (August 27, 2009)

All Versions

- Feature: Including both 32- and 64-bit libraries on 64-bit Linux release

- Feature: Now shipping precompiled Benchmark example on Linux builds

- Feature: Troubleshooting section added to Programmer's Guide

- Feature: Added scripts that report system information to *examples* folder

- Improved: Error output for examples is now more descriptive

- Improved: Documentation is more specific about configuring system runtime

- Fixed: Incompatibilities with gcc 4.2 and earlier; gcc 4.1 is now compatible

Basic

- Improved: gesvd was optimized for up to a 60% speedup over Beta 1

- Fixed: Error in geqrf for matrices of M << N

- Fixed: Error in gesvd where some matrices would yield non-unitary U and Vt

Premium

- Feature: Implemented getri

- Feature: Implemented potrf

- Feature: Implemented potrs

- Feature: Implemented posv

- Feature: Implemented trtrs

- Improved: orglq was optimized for up to a 700% speedup

9.6 Release 1.0 Beta 1 (August 13, 2009)

All Versions

- Feature: Support Windows XP 32/64

- Feature: Support Linux 32/64

Basic

- Feature: Implemented gels

- Feature: Implemented geqrf

- Feature: Implemented gesv

- Feature: Implemented gesvd
- Feature: Implemented getrf
- Feature: Implemented gglse

Premium

- Feature: Implemented gebrd
- Feature: Implemented getrs
- Feature: Implemented trtrs
- Feature: Implemented gelqf
- Feature: Implemented gerqf
- Feature: Implemented orgqr
- Feature: Implemented orglq
- Feature: Implemented orgbr
- Feature: Implemented ormqr
- Feature: Implemented ormlq
- Feature: Implemented ormrq
- Feature: Implemented bdsqr