PGI® CDK™ 5.2
Cluster Development Kit®

Installation
& Release Notes
5.2-2

The Portland Group™ Compiler Technology.
STMicroelectronics, Inc
9150 SW Pioneer Court, Suite H
Wilsonville, OR 97070
www.pgroup.com

PGI 5.2 Cluster Development Kit (CDK™)
Installation & Release Notes

First Printing: Release 5.2, August 2004

Technical support: trs@pgroup.com
http://www.pgroup.com

Table of Contents

TABLE OF CONTENTS.....I

1 PGI CDK 5.2 INTRODUCTION.....2

1.1 PRODUCT OVERVIEW.....3
1.2 TERMS AND DEFINITIONS.....5

2 PGI CDK 5.2 INSTALLATION NOTES.....8

2.1 INTRODUCTION.....9
2.2 INSTALLING ON LINUX86 OR LINUX86-64.....12
2.3 USING FLEXLM ON LINUX.....18

3 USING THE OPEN SOURCE CLUSTER UTILITIES.....23

3.1 RUNNING AN MPICH PROGRAM.....23
3.2 MORE ABOUT TORQUE.....26
3.3 LINKING WITH SCALAPACK.....27
3.4 TESTING AND BENCHMARKING.....28

4 PGI CDK 5.2 RELEASE NOTES.....30

4.1 PRODUCT CONTENTS.....30
4.2 SUPPORTED PRODUCTS.....32
4.2.1 Supported Processors.....32
4.2.2 Supported Operating Systems.....33
4.3 NEW COMPILER FEATURES.....36
4.4 COMPILER OPTIONS.....38
4.4.1 Getting Started.....38
4.4.2 New or Modified Compiler Options.....39
4.5 64-BIT SUPPORT.....41

4.5.1 Practical Limitations of –mcmodel=medium *44*
4.5.2 Compiler Limitations of –mcmodel=medium *45*
4.5.3 Large Array Example in C *46*
4.5.4 Large Array Example in Fortran *48*
4.6 PGDBG AND PGPROF 49
4.6.1 PGDBG AND PGPROF NEW FEATURES *50*
4.6.2 PGDBG AND PGPROF CORRECTIONS *51*
4.7 KNOWN LIMITATIONS 52
4.8 CORRECTIONS 54

5 PGHPF 5.2 60

5.1 SUMMARY OF CHANGES 60
5.2 RESTRICTIONS 61

6 CONTACTING PGI & ONLINE DOCUMENTATION 64

1 PGI CDK 5.2

Introduction

Welcome to *PGI Cluster Development Kit 5.2*, or *PGI CDK 5.2*, a set of Fortran, C and C++ compilers and development tools for 32-bit *x86*-compatible and 64-bit AMD64-compatible processor-based workstations and servers running versions of the Linux operating system.

*A **Cluster*** is a collection of compatible computers connected by a network. *The* PGI CDK Cluster Development Kit supports parallel computation on clusters of *Intel Pentium 2/3/4* or *AMD Athlon/AthlonXP/AthlonMP* compatible *Linux* workstations or servers interconnected by a *TCP/IP*-based network such as *Ethernet or Fast Ethernet.* With the *PGI CDK Release 5.2,* the 64-bit AMD Opteron and Xeon EM64T (AMD64 technology) processor-based systems are supported as well.

Note that support for cluster programming does not extend to clusters combining 64-bit AMD64 CPU-based systems with 32-bit CPU-based systems, unless all are running 32-bit applications built for a common set of working x86 instructions.

Release 5.2 is the second production release of *PGI CDK* that supports AMD64 technology processor-based systems, with large array addressing in *pgf77*, *pgf90*, and *pgcc*. These systems can utilize a 64-bit address space

while retaining the ability to run legacy 32-bit *x86* executables at full speed.

1.1 Product Overview

The *PGI CDK 5.2* product is comprised of the following components

- *PGF90* native OpenMP* and auto-parallelizing Fortran 95 compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments. Release 5.2 introduces full Fortran 95 and *large array* (single data objects larger than 2GB) support in *PGF90* for *linux86-64* environments.
- *PGF77* native OpenMP and auto-parallelizing FORTRAN 77 compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGCC* native OpenMP and auto-parallelizing ANSI and K&R *C* compiler in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGC++* native OpenMP and auto-parallelizing ANSI *C++* compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGPROF Multi-process/multi-threaded* graphical profiler in versions that will run on *linux86* and *linux86-64*, and a command-level version for *linux86*, *linux86-64*, and environments.
- *PGDBG Multi-process/multi-thread* graphical debugger, in versions that will run on *linux86* and *linux86-64* development environments.

- *MPICH MPI libraries, version 1.2.6,* for both 32-bit and 64-bit development environments.
- TORQUE 1.0.1 resource management system, a continuation of the product known as *Open PBS*, or Portable Batch System. See http://www.supercluster.org/projects/torque for more information.
- *ScaLAPACK* linear algebra math library for distributed-memory systems, including *BLACS* version 1.1 (the Basic Linear Algebra Communication Subroutines) and *ScaLAPACK* version 1.7 for use with *MPICH* and the PGI compilers on Linux systems with a kernel revision of 2.2.10 or higher. This is provided in both *linux86* and *linux86-64* versions for AMD64 CPU-based installations.

and the following documentation and tutorial materials:

- *OSC Training Materials* – an extensive set of HTML-based parallel and scientific programming training materials developed by the Ohio Supercomputer Center
- Complete online Documentation for the PGI compilers and tools in a mixture of HTML and PDF.
- Online HPF tutorials that provide insight into cluster programming considerations.
- Online Linux man pages for all of the supplied software
- A hard-copy CD-ROM media kit including the *PGI User's Guid*e, *MPI The Complete Reference, Volume 1*, the *High Performance Fortran Handbook*, *How to Build a Beowulf*, and a printed copy of these release notes.

Note that the compilers and libraries can be installed on other platforms not in the user cluster, as long as all platforms use a common floating license server.

1.2 Terms and Definitions

There are a number of terms used in this document that may be unfamiliar or used in an unfamiliar context. Following are definitions of terms used in the context of these *PGI CDK 5.2* release notes.

driver – the compiler *driver* controls the compiler, linker, and assembler and adds objects and libraries to create an executable. The *–dryrun* option illustrates operation of the driver. `pgf77`, `pgf90`, `pghpf`, `pgcc`, and `pgCC` are drivers for the PGI compilers.

x86 – a processor designed to be binary compatible with i386/i486 and previous generation processors from Intel* Corporation.

x87 – 80-bit IEEE floating-point unit (FPU) and associated instructions on *x8*6–compatible CPUs.

IA32 – an Intel Architecture 32-bit processor designed to be binary compatible with *x86* processors, but incorporating new features such as streaming SIMD extensions (SSE) for improved performance. This includes the Intel Pentium* 4 and Xeon* processors. For simplicity, these release notes refer to *x86* and *IA32* processors collectively as *32-bit x86* processors.

AMD64 – a 64-bit processor designed to be binary compatible with 32-bit *x86* processors, and incorporating new features such as additional registers and 64-bit addressing support for improved performance and greatly increased memory range. This includes the AMD* Athlon64* and Opteron* processors. Most comments in these release notes that apply to AMD64 technology processors also apply to IA32 processors with EM64T extensions.

SSE1 - 32-bit IEEE 754 FPU and associated *streaming SIMD extensions* (SSE) instructions on Pentium III, AthlonXP* and later 32-bit *x86* and *AMD64* compatible CPUs, enabling scalar and packed vector arithmetic on 32-bit floating-point data

SSE2 – 64-bit IEEE 754 FPU and associated SSE instructions on P4/Xeon

and later 32-bit *x86* and *AMD64* compatible CPUs, enabling scalar and packed vector arithmetic on 64-bit floating-point data

SSE3 – additional 32-bit and 64-bit SSE instructions to enable more efficient support of arithmetic on complex floating-point data on 32-bit *x86* and *AMD64* compatible CPUs with so-called *Prescott New Instructions* (PNI), such as the Intel Xeon EM64T.

linux86 – 32-bit Linux operating system running on an *x86* or *AMD64* processor-based system, with 32-bit GNU tools, utilities and libraries used by the *PGI CDK* compilers to assemble and link for execution.

linux86-64 – 64-bit Linux operating system running on an *AMD64* processor-based system, with 64-bit or 32-bit GNU tools, utilities and libraries used by the *PGI CDK* compilers to assemble and link for execution in either *linux86* or *linux86-64* environments. The 32-bit development tools and execution environment under *linux86-64* are considered a cross development environment for *x86* processor-based applications.
–mcmodel=small – Compiler/linker switch to produce *small memory model* format objects/executables in which both code (*.text*) and data *(.bss*) sections are limited to less than 2GB. This is the default (and only possible) format for *linux86* 32-bit executables. This is the default format for *linux86-64* executables. Maximum address offset range is 32-bits, and total memory used for OS+Code+Data must be less than 2GB

–mcmodel=medium – Compiler/linker switch to produce *medium memory model* format objects/executables in which code sections are limited to less than 2GB, but data sections can be greater than 2GB. *Not* supported in *linux86* 32-bit environments. Supported in *linux86-64* environments. This option must be used to *compile* any program unit that will be linked in to a 64-bit executable that will use aggregate data sets larger than 2GB and access data requiring address offsets greater than 2GB. This option must be used to *link* any 64-bit executable that will use aggregate data sets greater than 2GB in size. This option must be used in combination with *–Mlarge_arrays* to *compile* a program unit in which any single data object is greater than 2GB in size. Executables linked using *–mcmodel=medium* can incorporate objects compiled using *–mcmodel=small* as long as the *small* objects are from a shared library. There can be a performance

penalty associated with programs compiled and linked using *–mcmodel=medium*; this is a limitation related to how 64-bit addressing is specified by the *X86-64 Application Binary Interface*, not a PGI compiler limitation.

Large Arrays – Arrays with total size larger than 2GB**,** requiring the compiler to use 64-bit index arithmetic for accesses to elements of the array. Program units that use *Large Arrays* must be compiled using both the
–mcmodel=medium and *–Mlarge_arrays* options. If *–Mlarge_arrays* is *not* used, but *–mcmodel=medium* is used, *aggregate* data sets can be larger than 2GB but no single data object can exceed 2GB in size.
Shared library – A library of the form *libxxx.so* containing objects that are dynamically linked into a program at the time of execution. Objects in a shared library are compiled *–fpic,* for *position-independent code.* Most Linux system and PGI runtime libraries are provided as shared libraries. Object files compiled using the *–mcmodel=medium* option cannot be compiled *–fpic* and included in shared libraries. This is a limitation of the *X86-64 Application Binary Interface*, not a PGI compiler limitation. However, object files from shared libraries can be dynamically linked into executables linked using the *–mcmodel=medium* option.

Static linking – Using *–Bstatic* to ensure all objects are included in the generated executable at link time. Static linking causes objects from static library archives of the form *libxxx.a* to be linked in to your executable, rather than dynamically linking the corresponding *libxxx.so* shared library. Static linking of executables linked using the *–mcmodel=medium* option is not supported.

Hyperthreading (HT) – Some IA32 CPUs incorporate extra registers that allow 2 threads to run on a single CPU with improved performance for some tasks. This is called *hyperthreading*, and abbreviated *HT*. Some *linux86* and *linux86-64* environments treat IA32 CPUs with HT as though there were a 2nd *pseudo* CPU, even though there is only one physical CPU. Unless the Linux kernel is *hyperthread-aware*, the second thread of an *OpenMP* program will be assigned to the *pseudo* CPU, rather than a real second physical processor (if one exists in the system). *OpenMP* Programs can run very slowly if the second thread is not properly assigned.

2 PGI CDK 5.2 Installation Notes

A Cluster is a collection of compatible computers connected by a network. The *PGI CDK Cluster Development Kit* supports parallel computation on clusters of *Intel Pentium 2/3/4* or *AMD Athlon/AthlonXP/AthlonMP* compatible *Linux* workstations or servers interconnected by a *TCP/IP*-based network such as *Ethernet* or *Fast Ethernet.* With the *PGI CDK Release 5.2,* the *64-bit AMD Opteron* and *Intel Xeon EM64T* (AMD64 technology) processor-based systems are supported as well.

The PGI CDK release is installed on a working *cluster* – it is not the purpose of this product to create a cluster, or to troubleshoot one. The PGI CDK release can be installed on a single node, and the node can be treated as if it is a *cluster*.

Note that support for cluster programming does not extend to clusters combining AMD64 CPU-based systems with 32-bit CPU-based systems, unless all are running 32-bit applications built for a common set of working x86 instructions.

Release 5.2 is the second production release of *PGI CDK* that supports AMD64 technology processor-based systems, with large array addressing in *pgf77*, *pgf90*, and *pgcc*. These systems can utilize a 64-bit address space while retaining the ability to run legacy 32-bit x86 executables at full speed.

For multi-process programming, like message passing programs set to execute on a cluster, we provide both a 32-bit and a 64-bit set of MPICH libraries, built for adding information useful in the cluster debugger *PGDBG* and the cluster profiler *PGPROF,* as well as being the inter-process communication standard.

2.1 Introduction

You can view the online HTML interface to the *PGI CDK* using any web browser. Enter the following URL in a web browser:

```
file:/mnt/cdrom/index.htm
```

running on a Linux system with the *PGI CDK* CD-ROM inserted and mounted on the CD-ROM drive. If you do not know how to insert and mount a CD-ROM on your Linux system, see Step 3 below in section 2.2 or contact your system administrator.

Generally, clusters are configured with a "master" node from which jobs are launched and "slave" nodes that are used only for computation. Typically, the master node is accessible from the general-purpose or "public" network and shares a file system with the other computers on your network using NFS. The master node and all of the slave nodes are interconnected using a second "private" network that is only accessible from computers that are part of the cluster. There are two common cluster configurations:

1) The master node is used only for compilation and job submission, and only the slave nodes are used for computation.

2) All nodes are used for computation, including the master node

To use *MPICH* in the first configuration, *Open PBS (TORQUE)* should be installed. Otherwise, the master node will be used as one of the computation nodes by the `mpirun` command by default; it is possible to exclude the master node in the second configuration if `mpirun` is invoked with the `-nolocal` option (see the man page for `mpirun`). If you are using the first configuration, it is possible to install *MPICH* and run parallel MPI

or HPF jobs without installing any of the other components. However, if you will have multiple users running jobs on your cluster simultaneously, you will likely want to use *Open PBS* to ensure your cluster nodes are allocated and used efficiently.

Typically, a master node has two network cards to allow communication to the outside network as well as to the cluster nodes themselves, which may be on their own subnet. If this is the case on your cluster, then when the installation script prompts you for the name of the master node, you should use the name associated with the network card connected to the cluster nodes.

Also, it is important to note that in order for MPICH and *Open PBS* to run correctly, access from each node to every other node must be available via the 'rsh' or 'ssh' command. For example, if a 3-node cluster consists of a master and 2 slaves named *master*, *node1*, *node2*, then as a user you should be able to issue the commands:

```
% rsh master date
% rsh node2 date
```

or

```
% ssh master date
% ssh node2 date
```

From *node1*, and similarly from *node2* and *master*.

By default, all of the PGI compilers and tools will be installed on your system. You will select which of the open source components to install. At this point, you should have determined:

- Which *PGI CDK* open source components you will install (see above for a list of these)
- The hostnames of all the nodes that will be included in your cluster - you will need a list of these during the installation
- Whether the master node will be strictly a front-end for compilation, job launching, etc or whether it will participate as a

compute node

- If you are installing *Open PBS* (*TORQUE*), which users at your site will have *Open PBS* queue manager permissions (you will need their usernames)

- Whether the compute nodes can share files with the master node (this is *strongly* recommended)

Section 2.2 below describes how to install the PGI Fortran, C and C++ compilers and tools on Linux using the *installcdk* script from the *PGI CDK* CD-ROM. ***NOTE:*** *you must have root permissions to successfully execute the installcdk script.* You must install the software as specified in section 2.2 and then follow the instructions in section 2.3 for configuring and starting the FLEXlm license daemon.

The FLEXlm license daemon enables use of the PGI compilers and tools by any user on any system networked to the system on which the PGI software is installed. For example, users can compile, debug, and profile using the *PGI CDK* compilers and tools on any system on your general-purpose network, subject to the constraints on concurrent usage for the product you have purchased.

Sections 3.1 - 3.5 describe basic usage of the Open Source components of the *PGI CDK*, including *MPICH*, the *Open PBS* batch queuing system, *ScaLAPACK* libraries, and the example benchmark programs and tutorials.

For the first 60 days after your purchase, you may submit technical questions about the *PGI CDK* compilers and tools to the online problem reporting site found at http://www.pgroup.com/support/index.htm. If you have purchased PGI's Subscription Service, you will have access to service for an additional 12 months and will be notified by e-mail when maintenance releases occur and are available for electronic download and installation. Contact PGI at *sales@pgroup.com* for information about the PGI Subscription Service for the products you have purchased.

MPICH, *TORQUE*, and *ScaLAPACK*, are all open source software

packages that are *not* formally supported by PGI. All source code for these components is included on the *PGI CDK* CD-ROM in the `cdk` subdirectory. Each of these components has end-user and implementer documentation, generally in the form of printable postscript, along with the source code. Support for these products is generally provided by their respective user communities, which you can learn more about at the following URLs:

- *MPICH* - `http://www-unix.mcs.anl.gov/mpi/mpich` contains a wealth of information, including online documentation, tutorials, FAQ files, patch distributions, and information on how to submit bug reports to the *MPICH* developers.

- *TORQUE* - `http://www.supercluster.org/projects/torque` is the main site for the *Open PBS* product we integrate with the CDK.
- *ScaLAPACK*- `http://www.netlib.org/scalapack` contains FAQ files and current distributions of *ScaLAPACK*.

2.2 Installing on Linux86 or Linux86-64

For installation on an AMD64 technology processor-based system running a *linux86-64* execution and development environment, the *PGI CDK* installation script will attempt to install both the *linux86* version and *linux86-64* version of the compiler products requested. If the user specifies `/usr/pgi` as the base directory, for example,

Name of directory	Contents
`/usr/pgi/linux86/5.2/bin`	*linux86* versions of the compilers and tools
`/usr/pgi/linux86/5.2/lib`	*linux86* versions of the libraries,

	created on all platforms in the cluster.
`/usr/pgi/linux86/5.2/liblf`	*linux86*-only large-file-support *(-Mlfs*) versions of the libraries.
`/usr/pgi/linux86/5.2/include`	*linux86* versions of header files
`/usr/pgi/linux86-64/5.2/bin`	*linux86-64* versions of the compilers and tools
`/usr/pgi/linux86-64/5.2/lib`	*linux86-64* versions of the libraries, created on all platforms in the cluster. Not to be used for ***–mcmodel=medium*** development.
`/usr/pgi/linux86-64/5.2/libso`	*linux86-64* ***–fPIC*** libraries for ***–mcmodel=medium*** support
`/usr/pgi/linux86-64/5.2/include`	*linux86-64* versions of header files

When the install script installs the *linux86-64* versions on a supported AMD64 technology processor-based system running a *linux86-64* environment, the *linux86* versions will be installed as well in a separate area. The compilers and supporting components have the same names, and the environment you target by default (*linux86-64* or *linux86*) will depend on the version of the compiler that comes first in your path.

Bring up a shell command window on your system. The instructions below assume you are using *csh*, *sh*, *ksh*, or some compatible shell. Appropriate modifications will be necessary when setting environment variables if you are using a shell that is not compatible with one of these three.

Step 1 – Create the directory in which you wish to install the *PGI CDK Compilers* and *tools*. ***NOTE:*** *The installation directory you choose must be accessible from all nodes in the cluster using a uniform pathname.* In the example below, we assume `/usr/pgi` which is the default installation directory. However, installation can occur in any directory. Please make sure that the installation directory has the necessary ownership and permissions appropriate for your site by using the `chown` and `chmod` commands.

Set the environment variable `PGI` to the name of the installation directory. Assuming `csh`:

```
% setenv PGI /usr/pgi
```

Or, assuming `sh`, `ksh`, or `bash`:

```
% export PGI=/usr/pgi
```

Step 2 – All software should fit into 250 MB of disk space. If you wish to install all of the source code for the open source components of the PGI CDK, about 350 MB of disk space is required. If you are installing on a linux86-64 environment, add another 250MB of disk space. Verify that you have sufficient space on the disk where your installation directory will be located.

Step 3 – The *installcdk* script ***must*** be run to properly install the software, and you must have root permissions to execute this script. If you do not have root privileges, you will need to get help from your system administrator during the installation process. If you are updating a previous release, or wish to reinstall, it is a good practice to run *uninstallcdk* before running *installcdk.*

If you are installing from a CD-ROM and you're not sure how to access the CD-ROM drive on your system, check with your system administrator. Typically, you must insert the CD-ROM into the CD-ROM drive on the master node and issue the following command:

```
% mount /mnt/cdrom
```

while logged in as root to make the data on the CD-ROM accessible. Next, issue the following command from your root window on the master node:

```
% /mnt/cdrom/installcdk
```

***NOTE*:** If you have difficulty running this script, especially on a *Slackware Linux* system, check the permissions on `/dev/null`. Permissions should be set to `crw-rw-rw-`. If they are not, reset them using `chmod` and try running the install script again, or try '`mount -o`

exec /mnt/cdrom'. Otherwise, check and see whether the CDROM was mounted with 'noexec' set. If so, unmount it and then mount it with 'mount –o exec /mnt/cdrom'

The install script will install all of the binaries for the PGI compilers and tools, *MPICH*, and *ScaLAPACK* in the $PGI directory tree in the appropriate bin, include, lib, and man subdirectories. It will also install *Open PBS* in the /usr/local/pbs and /var/spool/pbs directories, and start the *Open PBS* daemons running on the master node and all of the compute nodes in your cluster. You will be prompted for various information about how to configure your cluster as the script executes. Once the installation script has completed, exit the root login.

Step 4 – All of the *PGI CDK* compilers and development tools are license-managed. The other components of the *PGI CDK*, including *MPICH*, *ScaLAPACK*, and the *Open PBS* batch scheduler, are open source products that are not license-managed.

If you choose to create a temporary demo license for the PGI compilers and tools, the install script asks for your real name, your username, and your email address. It then creates a fifteen-day license and prints a message like this:

```
NOTE: your evaluation license will expire in
14 days, 23.6 hours. For a permanent license,
please read the order acknowledgement that you
received.  Connect to https://www.pgroup.com/License
with the username and password in the order
acknowledgement.


Name:   <your name>
User:   <your username>
Email:  <your e-mail address>
Hostid: PGI=9BF378E0131FF0C3CD37F6
FLEXlm hostid: 00a024a3dfe7
Hostname: yourhost.yourdomain.com
Installation: /usr/pgi
PGI Release: 5.2-2
```

The message above is also saved to the file $PGI/license.info for

retrieval at a later time.

Once you have obtained your permanent license keys using your personalized account on the PGI web page, place them in the file `$PGI/license.dat`. Until the permanent license is obtained, the *PGI CDK* compilers will only be usable under the username specified above during generation of the temporary keys.

Step 5 – You can view the online HTML interface to the *PGI CDK* documentation and tutorial materials using any web browser. Assuming you use *Netscape*, issue the following command:

```
% netscape $PGI/index.htm
```

You can view the online HTML manuals for the PGI compilers and tools directly by issuing the following command:

```
% netscape $PGI/doc/index.htm
```

You may want to place a bookmark on these locations for easy future reference to the online manuals.

Step 6 – Once the temporary or permanent license file is in place, execute the following commands to make the *PGI CDK* compilers accessible from a normal user shell window.

Assuming `csh`, for *linux86* executable development tools:

```
% set path = ( usr/pgi/linux86/5.2/bin\
  /usr/local/pbs/bin $path )
```

alternatively, for *linux86-64* executable development tools:

```
% set path = ( usr/pgi/linux86-64/5.2/bin\
  /usr/local/pbs/bin $path )
```

and for the common documentation:

```
% setenv MANPATH "$MANPATH":/usr/pgi/linux86/5.2/man:\
  /usr/local/pbs/man
```

Or, assuming `sh` or `ksh`, for *linux86* executable development tools:

```
% export PATH=\
 /usr/pgi/linux86/5.2/bin:/usr/local/pbs/bin:$PATH
```

alternatively, for *linux86-64* executable development tools:

```
% export PATH=\
 /usr/pgi/linux86-64/5.2/bin:/usr/local/pbs/bin:$PATH
```

and for the common documentation:

```
% export MANPATH=\
$MANPATH:/usr/pgi/linux86/5.2/man:/usr/local/pbs/man
```

Note: you may see a warning message when you set the `MANPATH` environment variable if you do not already have `MANPATH` defined.

Each person who will be using the *PGI CDK* Fortran, C, and C++ compilers and tools should add the above commands to his or her startup files (e.g. `.cshrc`) to enable access to the PGI compilers and tools by default upon future logins. For license support (see 2.3) you might also want to add the environment variable declarations for **$PGI** and **$LM_LICENSE_FILE**

Step 7 – You can verify the release number of the compilers you have installed using the *–dryrun –V* options on any of the compiler commands. This will also show you the sequence of steps the compiler will use to compile and link programs for execution on your system.

- For Fortran 77, use "`pgf77 -V x.f`"
- For Fortran 90, use "`pgf90 -V x.f`"
- For HPF, use "`pghpf -V x.f`"
- For C++, use "`pgCC -V x.c`"
- For C, use "`pgcc -V x.c`"

Note that the files `x.f` or `x.c` need not exist in order for you to successfully execute these commands to determine the release number, although it will correctly report a missing source file.

The base installation of the *PGI CDK* is now complete. Follow the directions in section 2.3 to enable floating license capability, and the following sections for basic usage instructions for *MPICH*, *Open PBS*, and *ScaLAPACK*.

2.3 Using FLEXlm on Linux

The *PGI CDK* Fortran, C, and C++ compilers and tools are license-managed using the FLEXlm software license management system from *Macrovision Inc*. The steps below describe how to set up and run license daemons on the master node of your cluster. These daemons allow the *PGI CDK* compilers to be used on any machine on your public network, subject to the constraint on maximum number of concurrent users as specified in your license. The master node will function as the license "server," distributing seats as requested for use on other computers on your network.

Alternatively, if you already use other software products managed using FLEXlm, you can incorporate the PGI products into the configuration you use for license serving those products.

Step 1 – Install the *PGI CDK* compilers as described in section 2.2 above.

Step 2 – Once you have obtained permanent license keys (see section 2.2 above for how to obtain these), place them in a file named `license.dat` in the `$PGI` directory. For example, the `license.dat` file should look similar to the following:

```
SERVER <hostname> <hostid> 7496
DAEMON pgroupd <install_dir>/linux86/5.2/bin/pgroupd

FEATURE pghpf-linux86 pgroupd 5.200 31-dec-0 1 \
2B9CF0F163159E4ABE32 VENDOR_STRING=123456:16:cdk \
```

```
HOSTID=<hostid> ck=49

FEATURE pghpf-linux86-64 pgroupd 5.200 31-dec-0 1 \
2B9CF0F163159E4ABE32 VENDOR_STRING=123456:16:cdk \
HOSTID=<hostid> ck=49
```

It will include features for each of the *PGI CDK* compilers and tools. `<hostname>` and `<hostid>` should match those you submitted to PGI and `<install_dir>` must be changed to match the directory in which the compilers are installed. In particular, `<install_dir>` should match the value of `$PGI` as defined above. Note that this example contains an entry for 32-bit *pghpf* product and one for the 64-bit *pghpf* product. If you have *linux86-64* products, make sure the license you generate has those entries as well.

Note: Release 5.2 supports newer versions of FLEXlm daemons. These new versions require all license features to be case-insensitive. We now have a license feature called ***pgcpp*** instead of ***pgCC***, and so older releases of ***pgCC*** compilers will not be compatible with the new **5.2** license. Contact ***trs@pgroup.com*** if you wish to have the **5.2** license work with **4.1** and **5.0** versions of ***pgCC***.

Step 4 – When the license file is in place, execute the following commands to make the PGI products you have purchased accessible. If you are *not* using other products managed by FLEXlm, and have *not* previously set the environment variable `LM_LICENSE_FILE`, issue the following command to do so (assuming `csh`):

```
% setenv PGI /usr/pgi
% setenv LM_LICENSE_FILE $PGI/license.dat
```

Or, assuming `sh`, `ksh`, or `bash`:

```
% export PGI=/usr/pgi
% export LM_LICENSE_FILE=$PGI/license.dat
```

If you *are* using other products managed by FLEXlm, and *have* previously set the environment variable `LM_LICENSE_FILE`, either incorporate the PGI license keys into your existing license file or issue the following command to append the PGI license file to the definition of

LM_LICENSE_FILE (assuming csh):

```
% setenv LM_LICENSE_FILE \
"$LM_LICENSE_FILE":$PGI/license.dat
```

Or, assuming sh or ksh or bash:

```
% export LM_LICENSE_FILE= \
$LM_LICENSE_FILE:$PGI/license.dat
```

Each person who will be using the *PGI CDK* compilers should add the above commands to his or her startup files to enable access to the PGI compilers and tools by default upon future logins.

NOTE: If LM_LICENSE_FILE is not set or exported, and the node-locked 15-day temporary license file $PGI/PGIinstall still exists, then $PGI/PGIinstall will be used for resolving compiler licenses.

Step 5 – You must now start the license manager daemon. Edit the shell script template $PGI/linux86/5.2/bin/lmgrd.rc. If you have installed the *PGI CDK* compilers in a directory other than /usr/pgi, substitute the correct installation directory into the definition of the PGI environment variable on line 3 of the script. Now exit the editor and issue the following command to start the license server and PGI license daemon running on your system:

```
% lmgrd.rc start
```

If you wish to stop the license server and license daemon at a later time, you can do so with the command:

```
% lmgrd.rc stop
```

To make sure that the license server and PGI daemon are started each time your system is booted, log in as root, set the PGI environment variable as above, and then execute the following two commands:

```
% cp $PGI/linux86/5.2/bin/lmgrd.rc
/etc/rc.d/init.d/lmgrd
```

```
% ln -s /etc/rc.d/init.d/lmgrd /etc/rc.d/rc3.d/S90lmgrd
```

Note that your system's default runlevel may be something other than '3', and if it is, that number should be used above in setting the correct subdirectory. Run `/sbin/runlevel` to check the system's runlevel. Note also that if you're using a Linux distribution other than Red Hat, your `rc` files may be in a directory other than `/etc/rc.d`. Some Linux distributions, such as Red Hat and Mandrake, include the *chkconfig(8)* utility that manages the runlevel scripts. If your system has this tool and you wish to use it, then run the following commands:

```
% cp $PGI/linux86/5.2/bin/lmgrd.rc /etc/rc.d/init.d
% chkconfig --add lmgrd.rc
```

The appropriate links will be created in the `/etc/rc.d` directory hierarchy. For more information on *chkconfig*, please see the manual page.

In addition to the `installcdk` installation script, there are two additional shell scripts that may be of use to you:

- `uninstallcdk` - uninstalls selected components of the *PGI CDK*. This is useful, for example, if you wish to download an updated version of one of the open source components and rebuild and reinstall it on your cluster.

- `restartpbs` - shuts down and restarts all of the *Open PBS* daemons on each node of your cluster. If you find that your default queue is not operating properly for some reason, try restarting *TORQUE* using this script to see if it fixes the problem.

As with the `installcdk` script, you must have root privileges in order to execute these scripts.

Installation of your *PGI CDK* Fortran, C, and C++ compilers and tools for Linux is now complete. If you have difficulties with the installation, send e-mail to *trs@pgroup.com* for assistance.

The following sections describe basic usage of the open source *PGI CDK*

clustering utilities.

3 Using the Open Source Cluster Utilities

Copy the directory `$PGI/bench` to a local working area so you can try an example program.

3.1 Running an MPICH Program

NOTE: you must either work in a directory which is file-shared with all of the cluster nodes, or you must copy your MPI executables to a common directory on all compute nodes before invocation of *mpirun*. In particular, this precludes you from working in `/tmp` unless you copy the executable to `/tmp` on each slave node prior to invocation of *mpirun*.

First, try the MPI "hello world" program in the `bench/mpihello` subdirectory:

```
% cp -r $PGI/bench ./bench
% cd ./bench/mpihello
% pgf77 -o mpihello mpihello.f -Mmpi
% mpirun mpihello
Hello world!  I'm node 0
% mpirun -np 4 mpihello
Hello world!  I'm node 0
Hello world!  I'm node 2
Hello world!  I'm node 1
Hello world!  I'm node 3
```

If you've installed *Open PBS (TORQUE)*, you should also try submitting a batch job to make sure your *Open PBS* default queue is operational. There is an example *Open PBS* batch script for submission of the above "hello world" program in the `bench/mpihello` subdirectory. It assumes you have a cluster with 4 or more processors. You'll need to modify the batch script, `mpihello.pbs`, to ensure the pathname information included is correct for your particular installation.

IMPORTANT Open PBS NOTE 1

*A batch job submitted using the Open PBS **qsub** command does not by default inherit the environment of the spawning shell. Instead, Open PBS batch jobs execute in an environment that is initialized based on the submitting user's login/shell startup files. It may be the case that a user's home directory and shell/startup files are not accessible from the slave nodes (which generally are on their own private network). **In this case, you must be sure that each end-user of Open PBS has a valid home directory in the /etc/passwd file on each cluster node** (generally it is the sixth field of a login entry in /etc/passwd). If the home directory entry for a given user on any node is invalid, Open PBS jobs will quietly fail in ways that are difficult to diagnose.*

IMPORTANT Open PBS NOTE 2

*In order for a Open PBS batch job to find the **mpirun** command, the necessary path initialization must be performed. It is best to perform the initialization either in each user's login/shell startup files as noted above, or in /etc on each slave node if you want to initialize it globally. If you aren't sure how to do this, contact your system administrator. Alternatively, you can use the -v or -V options to **qsub** (see the **qsub** man page for more on these options) to pass environment variables to the submitted job's environment, or you can explicitly initialize the path environment variable within the Open PBS batch script. The latter method is used in the example below. If the environment of a given batch job is not properly initialized in one of these ways, Open PBS jobs can fail to execute in ways that are difficult to diagnose.*

Now, try submitting a *Open PBS* batch job using the following command:

```
% qsub mpihello.pbs
% qstat
```

You'll need to type `qstat` quickly in order to see the "mpihello" job in the queue. Be sure to look at the `mpihello.log` file when the job completes to see that the job has executed correctly. You should see output something like the following:

```
% cat mpihello.log
Hello world!  I'm node 0
Hello world!  I'm node 2
Hello world!  I'm node 1
Hello world!  I'm node 3
%
```

If these simple tests don't work, refer to the *IMPORTANT Open PBS NOTES* above. Usually, it's either a problem with

- A user not having a valid home directory entry in `/etc/passwd` on each cluster node
- An incorrectly initialized `PATH` variable in the shells executing the *Open PBS* job
- Inability of one or more of the slave nodes to find "mpirun" because the PGI software has been installed in a directory which is not visible to them

If these simple tests do work, you're ready to execute some of the more extensive tests listed below in section 3.4, *Testing and Benchmarking*.

The following sections include more detailed information on each of the open source components of the PGI CDK.

3.2 More About TORQUE

The `installcdk` script installs *Open PBS (TORQUE)* for a space-shared cluster; that is, multiple jobs can be run at the same time, but at most one job will use a given node at any given time. Optionally, one node can be designated as a front-end node (usually call the master node) that can be used to submit jobs. Many more options are available; to learn more, read the *Open PBS Administration Guide*, which is found in the `cdk/pbs` subdirectory of the *PGI CDK* CD-ROM.

Check that the *Open PBS* man pages are properly installed by bringing up one of the man pages:

```
% export MANPATH=/usr/local/pbs/man:$MANPATH
% man qsub
```

NOTE: you may have to stop and then re-start the default queue once after *Open PBS* installation is complete and prior to running your first job. It's not clear why this is necessary, but if your test job seems to queue up and not run (you can check its status using the qstat command), try issuing the following commands:

```
% qstop default
% qstart default
```

and then re-submitting the job. You must be registered as a *Open PBS* queue manager or be logged in as root to execute these commands. If you need to add queue managers at a later time, you may do so using the "qmgr" command while logged in as root:

```
% qmgr
Qmgr: set server managers=<username>@<masternode>
Qmgr: quit
```

where <username> is replaced with the username of the person who will become a queue manager, and <masternode> is replaced with the simple hostname of the master node in your cluster. ***NOTE:*** *In this case, the hostname cannot be a full hostname. That is, if the full hostname of your master node is* `*.pgroup.com`*, you would enter* `*` *in place of* `<masternode>` *in the* `set server` *command.*

3.3 Linking with ScaLAPACK

The *ScaLAPACK* libraries are automatically installed as part of step 1 above. You can link with the *ScaLAPACK* libraries by specifying *–Mscalapack* on any of the *PGI CDK* compiler command lines. For example:

```
% pgf77 myprog.f -Mscalapack
```

The -Mscalapack option causes the following libraries, all of which are installed in $PGI/linux86/5.2/lib, to be linked in to your executable:

- `scalapack.a`
- `blacsCinit_MPI-LINUX-0.a`
- `blacs_MPI-LINUX-0.a`
- `blacsF77init_MPI-LINUX-0.a`
- `libblas.a`
- `libmpich.a`

You can run a program that uses *ScaLAPACK* routines just like any other MPI program. The version of *ScaLAPACK* included in the *PGI CDK* is pre-configured for use with *MPICH*. If you wish to use a different BLAS library, and still use the *–Mscalapack* switch, you will have to copy your BLAS library into `$PGI/linux86/5.2/lib/libblas.a`.

Alternatively, you can just list the above set of libraries explicitly on your

link line. You can test that *ScaLAPACK* is properly installed by running a test program as outlined below in section 3.4, *Testing and Benchmarking*.

3.4 Testing and Benchmarking

The directory `bench` on the *PGI CDK* CD-ROM contains various benchmarks and tests. Copy this directory into a local working directory by issuing the following command:

```
% cp -r $PGI/bench .
```

NAS Parallel Benchmarks - The `NPB2.3` subdirectory contains version 2.3 of the NAS Parallel Benchmarks in MPI. Issue the following commands to run the BT benchmark on 4 nodes of your cluster:

```
% cd bench/NPB2.3/BT
% make BT NPROCS=4 CLASS=W
% cd ../bin
% mpirun -np 4 bt.W.4
```

There are several other NAS parallel benchmarks available in this directory. Similar commands are used to build and run each of them. Try building the Class A version of BT if you'd like to run a larger problem (just substitute "A" for "W" in the commands above).

The example above runs the BT benchmark on 4 nodes, but does not use the *Open PBS* batch queuing system. There is a pre-configured *Open PBS* batch file in the `NPB2.3/bin` sub-directory. Edit the file, change the `cd` command in the second to last line of the script to point to your local working directory, and then try executing the following commands to run BT under control of *Open PBS*:

```
% cd bin
% qsub bt.pbs
```

You can check on the status of your job using the qstat command.

The `hpfnpb` subdirectory contains versions of five of the *NAS Parallel Benchmarks* coded in High Performance Fortran (HPF). README files explain how to build and run each of these benchmarks on various platforms. Use the instructions and makefiles in the `linux86` subdirectories of each benchmark to test these programs on your cluster.

MPICH - These tests measure latency and bandwidth of your cluster interconnect for MPICH messaging. To run these tests, execute the following commands:

```
% cd mpi
% make
% mpirun -np 2 mpptest
```

For more information, the `runmpptest` script can be executed. PGI has noted significant latency increases on Linux when messages larger than about 7600 bytes are sent, so this script may take some time to run. See the `mpich/examples/perftest` directory for more information.

ScaLAPACK - This test will time execution of the 3D PBLAS (parallel BLAS) on your cluster:

```
% cd scalapack
% make
% mpirun -np 4 pdbla3tim
```

Matrix Multiplication - This test will time execution of a simple distributed matrix multiply on your cluster:

```
% cd matmul
% buildhpf
% mpirun -np 4 matmul_hpf
```

4 PGI CDK 5.2 Release Notes

This document describes changes between *PGI CDK 5.2* and previous releases, as well as information not included in the current printing of the *PGI User's Guide*. See also http://www.pgroup.com/support/new_rel.htm for the most recent information not included here.

There are two versions of *PGI CDK 5.2*:

- A *32-bit* version supported on 32-bit operating systems running on either a 32-bit x86 compatible or a 64-bit AMD64 compatible processor

- A *64-bit/32-bit* version that includes all features and capabilities of the 32-bit version, and which is also supported on 64-bit operating systems running on 64-bit AMD64 compatible processors.

These versions are distinguished in these release notes where necessary.

4.1 Product Contents

The *PGI CDK 5.2* product is comprised of the following components

- *PGF90* native OpenMP* and auto-parallelizing Fortran 95 compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.

Release 5.2 introduces full Fortran 95 and *large array* (single data objects larger than 2GB) support in *PGF90* for *linux86-64* environments.

- *PGF77* native OpenMP and auto-parallelizing FORTRAN 77 compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGCC* native OpenMP and auto-parallelizing ANSI and K&R *C* compiler in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGC++* native OpenMP and auto-parallelizing ANSI *C++* compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGPROF Multi-process/multi-threaded* graphical profiler in versions that will run on *linux86* and *linux86-64*, and a command-level version for *linux86*, *linux86-64*, and environments.
- *PGDBG Multi-process/multi-thread* graphical debugger, in versions that will run on linux86 and *linux86-64* development environments.
- *MPICH MPI libraries, version 1.2.6,* for both 32-bit and 64-bit development environments.
- TORQUE 1.0.1 resource management system, a continuation of the product known as *Open PBS*, or *Portable Batch System*. See http://www.supercluster.org/projects/torque for more information.
- *ScaLAPACK* linear algebra math library for distributed-memory systems, including *BLACS* version 1.1 (the Basic Linear Algebra Communication Subroutines) and *ScaLAPACK* version

1.7 for use with *MPICH* and the PGI compilers on Linux systems with a kernel revision of 2.2.10 or higher. This is provided in both *linux86* and *linux86-64* versions for AMD64 CPU-based installations.

and the following documentation and tutorial materials:

- *OSC Training Materials* – an extensive set of HTML-based parallel and scientific programming training materials developed by the Ohio Supercomputer Center
- Complete online Documentation for the PGI compilers and tools in a mixture of HTML and PDF.
- Online HPF tutorials that provide insight into cluster programming considerations.
- Online Linux man pages for all of the supplied software
- A hard-copy CD-ROM media kit including the *PGI User's Guid*e, *MPI The Complete Reference, Volume 1*, the *High Performance Fortran Handbook*, *How to Build a Beowulf*, and a printed copy of these release notes.

Note that the compilers and libraries can be installed on other platforms not in the user cluster, as long as all platforms use a common floating license server.

4.2 Supported Products

4.2.1 Supported Processors

PGI CDK 5.2 is supported on the following processors. The *–tp <target>* command-line option is used to generate executables that utilize features and optimizations specific to a given CPU and operating system environment. Compilers included in a 64-bit/32-bit *PGI CDK 5.2* installation can produce executables targeted to any 64-bit or 32-bit target,

including cross-targeting for AMD and Intel 64-bit AMD64 compatible CPUs. Compilers included in a 32-bit *PGI CDK 5.2* installation *cannot* produce executables for 64-bit targets. The default, in the absence of an explicit *–tp* switch, is for the PGI compilers to produce executables targeted to the CPU and operating system environment on which compilation is performed.

***PGI CDK 5.2* Supported Processors**							
Supplier	CPU	<*target*>	Memory Address		Floating Point HW		
			32-bit x86	64-bit x86-64	80-bit x87	32-bit SSE1	64-bit SSE2
AMD	Opteron/Athlon64	k8-64	No	Yes	Yes	Yes	Yes
AMD	Opteron/Athlon64	k8-32	Yes	No	Yes	Yes	Yes
Intel	Xeon EM64T	p7-64	No	Yes	Yes	Yes	Yes
Intel	Xeon EM64T	p7	Yes	No	Yes	Yes	Yes
Intel	Xeon/Pentium4	p7	Yes	No	Yes	Yes	Yes
AMD	Athlon XP/MP	athlonxp	Yes	No	Yes	Yes	No
Intel	Pentium III	piii	Yes	No	Yes	Yes	No
AMD	Athlon	athlon	Yes	No	Yes	No	No
AMD	K6	k6	Yes	No	Yes	No	No
Intel	Pentium II	p6	Yes	No	Yes	No	No
Various	Other x86	p5 or px	Yes	No	Yes	No	No

NOTE: The Intel Xeon EM64T supports new floating-point hardware instructions known as SSE3. The *PGI CDK 5.2* compilers make no use of these instructions, even in the presence of the *–tp p7-64* command-line option. However, optimized code for a Xeon EM64T processor is different from code compiled for an Opteron processor, because of performance differences of code sequences on different processors.

4.2.2 Supported Operating Systems

PGI CDK 5.2 is supported on the operating systems listed in the table below, and their equivalents. To determine if *PGI CDK 5.2* will install and

run under a Linux equivalent version (*Mandrake*, Debian*, Gentoo**, etc), look to see if a supported system with the same *glibc* and *gcc* versions is in the table. Other version differences can cause difficulties, but often these can be overcome with minor adjustments to the PGI software installation or operating system environment.

Newer distributions of the Linux operating system include support for 64-bit AMD64 compatible processors (AMD Athlon64/Opteron, Intel Xeon EM64T), and are designated *64-bit* in the table. These are the only distributions on which the 64-bit/32-bit version of *PGI CDK 5.2* will fully install. The MPICH libraries were built specifically for the 64-bit OS versions listed, and other versions may not install successfully.

If you attempt to install the 64-bit/32-bit version of *PGI CDK 5.2* on a system running a 32-bit Linux distribution, only the 32-bit versions of the PGI compilers and tools will be installed.

Some newer Linux distributions support the *Native Posix Threads Library* (NPTL), a new threads library that can be utilized in place of the *libpthreads* library available in earlier versions of Linux. Distributions that include NPTL are designated in the table. Parallel executables generated using the *OpenMP* and auto-parallelization features of the *PGI CDK 5.2* compilers will automatically make use of NPTL on distributions where it is available. In addition, the *PGDBG* debugger is capable of debugging executables built using either NPTL or *libpthreads*.

Operating Systems Supported by PGI CDK 5.2							
Distribution	**Type**	**64-bit**	**HT**	**PGC++**	**PGDBG**	**NPTL**	**glibc**
RHEL 3.0	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.2
RHEL 2.9.5	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.2
Fedora C-2	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.2
SuSE* 9.1	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.3
SuSE 9.0	***Linux***	Yes	Yes	Yes	Yes	No	2.3.2
SLES 8 SP2	***Linux***	Yes	Yes	Yes	Yes	No	2.2.5
SLES 9.0	***Linux***	Yes	Yes	Yes	Yes	Yes	2.3.3
SuSE 8.2	***Linux***	Yes	Yes	Yes	Yes	No	2.3.2
SuSE 8.1	***Linux***	Yes	Yes	Yes	Yes	No	2.2.5
SuSE 8.0	***Linux***	No	No	Yes	Yes	No	2.2.5
SuSE 7.3	***Linux***	No	No	Yes	Yes	No	2.2.4
SuSE 7.2	***Linux***	No	No	Yes	Yes	No	2.2.4
SuSE 7.1	***Linux***	No	No	Yes	Yes	No	2.2.4
Red Hat* 9.0	***Linux***	No	No	Yes	Yes	Yes	2.3.2
Red Hat 8.0	***Linux***	No	No	Yes	Yes	No	2.2.93
Red Hat 7.3	***Linux***	No	No	Yes	Yes	No	2.2.5
Red Hat 7.2	***Linux***	No	No	Yes	Yes	No	2.2.4
Red Hat 7.1	***Linux***	No	No	Yes	Yes	No	2.2.3
Red Hat 7.0	***Linux***	No	No	Yes	Yes	No	2.2

Either the 32-bit-only or the 64-bit/32-bit version of *PGI CDK 5.2* can be installed and function correctly on any 64-bit Linux distribution. However,

- For the 64-bit versions of the PGI compilers to function correctly, *gcc* must be installed and configured to support 64-bit programming. For example,

```
gcc -mcmodel=medium x.c
```

should compile and execute without incident, where `x.c` is any valid C program that addresses large data sets.

- For the 32-bit versions of the PGI compilers to function correctly, *gcc* must be installed and configured to support 32-bit programming. For example,

```
gcc -m32 hello.c
```

should compile and execute correctly. If *gcc* is *not* configured for 32-bit support, the PGI 32-bit *linux86* compilers will *NOT* install correctly on a 64-bit Linux distribution.

For *OpenMP* programming, machines with 2 or more physical CPUs with hyperthreading (HT) capability enabled must run a Linux distribution that schedules threads in a way that favors the 2nd physical CPU over the current CPU's 2nd *pseudo* CPU. These distributions are designated in the HT Column. If you intend to run multi-process *OpenMP* programs on a system with HT-capable processors without an HT-aware Linux distribution, you should disable the HT feature of the processors in the system.

NOTE: While *SuSE Linux Enterprise Server 8* and *SuSE 8.1* are 64-bit Linux distributions, there are known limitations to the default versions of the GNU tools (*gcc*, assembler, linker) on these distributions that prevent *Large Arrays* support. If you are installing the *PGI CDK 5.2* compilers and tools on one of these distributions, you may need to upgrade to a newer version of the GNU tools.

NOTE: *http://www.pgroup.com/support/install.htm* lists any new Linux distributions that may be explicitly supported by the *PGI CDK 5.2* compilers. If your Linux distribution is newer than any of those listed in the table above, the installation may still be successful. The web page *http://www.pgroup.com/support/install.htm* covers many common online license key generation questions.

4.5 New Compiler Features

Following are the new features of the *PGI CDK 5.2* compilers and tools as compared to prior releases.

- *Fortran 95* – the *PGF90* compiler now supports full Fortran 95. The command name `pgf90` is retained to enable full backward compatibility of build environments with previous releases of the PGI compilers and tools.
- *Large Arrays* – single data objects larger than 2GB in size are

now supported by *PGF90* in *linux86-64* environments. Data objects, both locally and globally declared, including objects in COMMON blocks and objects allocated on the stack, can now be larger than 2GB individually and as an aggregate. The *–Mlarge_arrays* option must be used with *–mcmodel=medium* to compile Fortran programs that use *Large Arrays*. The *–Mlarge_arrays* option will become default in *linux86-64* environments for *PGF90*, *PGF77* and *PGCC* in a future release of the PGI compilers and tools.

- *One-pass IPA* – the inter-procedural analysis (IPA) and optimization phases of the PGI compilers have been re-worked to enable *one-pass IPA*, meaning that *–Mipa* can be used on the compiler command lines like any other compiler option with no required changes to make files or build scripts. Previous releases of the PGI compilers and tools used a two-pass IPA implementation.
- *IPA-driven function inlining* – the IPA phase of the PGI compilers now aggressively inlines functions in the presence of the *–Mipa=inline* option. While this inlining is not recommended in all cases, it can improve performance of applications that rely on large numbers of calls to relatively small functions. Inlining can also increase opportunities for important loop optimizations, such as loop unrolling and vectorization.
- *Performance* – Further tuning of both the 32-bit x86 and 64-bit AMD64 code generators and other optimization phases, resulting in *SPECFP2K* Fortran performance improvements averaging 10% over the previous *PGI CDK 5.1* release. Several important research community applications like *MM5*, *MOLPRO*, *GAMESS*, *WRF* and *POP* show performance increases of 10% to 20%. C performance is improved modestly, typically by 5% to 10% on several industry-standard benchmarks.
- *64-bit integer optimizations* – full optimization of loops that require 64-bit index variables on *AMD64* and compatible

processors running a *linux86-64* environment, including for example loops in programs that are compiled using the *–i8* option.

- *Prefetch directives – PGF77* and *PGF90* now support directives and *PGCC* supports pragmas to allow explicit prefetching of data by the programmer. See the *PGI User's Guide* for details on how to use these directives and pragmas.
- *ACML 2.1* – a new edition of the *AMD Core Math Library, ACML 2.1*, is bundled with the *PGI CDK 5.2* compilers and tools. The bundled version of the ACML supports only 32-bit *x86* and 64-bit *AMD64* compatible CPUs that support both SSE1 and SSE2 instructions. The lower-performance but fully portable *libblas.a* and *liblapack.a* libraries are still included, and can be used on CPUs that do not support SSE instructions.
- *EM64T support* – Intel IA32 processors with EM64T extensions, designed to be binary compatible with AMD64 technology processors from AMD, are now supported using the *–tp p7-64* command-line option.
- *Expanded operating systems support* – several new Linux distributions are supported, including *SuSE 9.1* and *Fedora Core 2*.

4.4 Compiler Options

4.4.1 Getting Started

By default, the *PGI CDK 5.2* compilers generate code optimized for the type of processor on which compilation is performed (the compilation host). If you are unfamiliar with the PGI compilers and tools, a good option to use by default is *–fast*. This option is host-dependent but usually includes the options *–O2 –Munroll –Mnoframe*. Typically, for best performance on processors that support SSE instructions, you will want to

use the *PGF90* compiler (even for FORTRAN 77 code) and the *–fastsse* option. This option is similar to *–fast*, but incorporates additional optimization options to enable use of streaming SIMD (SSE/SSE2) instructions where appropriate. The contents of the *–fastsse* switch are host-dependent, but typically include the options *–O2 –Munroll –Mnoframe –Mlre –Mvect=sse –Mcache_align*. On some systems, *–fastsse* also includes *–Mscalarsse* and *–Mflushz*.

In addition to *–fastsse*, the *–Mipa=fast* option for inter-procedural analysis and optimization can improve performance. You may be able to obtain further performance improvements by experimenting with the individual *–Mpgflag* options detailed in the *PGI User's Guide* (*–Mvect*, *–Munroll*, *–Minline*, *–Mconcur*, etc). However, speed-ups using these options are typically application and system-dependent, so it is important to time your application carefully when using these options to ensure no performance degradations occur.

4.4.2 New or Modified Compiler Options

The following compiler options have been added or modified in *PGI CDK Release 5.2*:

- *–Bdynamic* – explicitly request that the compiler drivers link with shared object libraries.

- *–Mipa=inline[:n]* – IPA-driven inlining has been significantly enhanced, and can provide performance improvements on some benchmarks and applications. If the optional *n* is supplied, perform inlining up to *n* levels from leaf routines upward. The default value for *n* is 3.

- *–Mlarge_arrays* – now applies to both the *PGF90* and *PGF77* compilers. This option must be used, in combination with *–mcmodel=medium*, when compiling F95 or F77 applications that use single data objects larger than 2GB in size.

- *–M[no]prefetch* – (disables) enables generation of prefetch

instructions; only applies when used in combination with *–Mvect* or an aggregate option such as *–fastsse* that incorporates *–Mvect*.

- *–Munsafe_par_align* – assume aligned moves are safe for array references in parallelized loops as long as the first element of the array is aligned. *NOTE*: this option will cause aligned moves to be generated even when the compiler can't prove they are safe. This can improve performance on some benchmarks, in particular the *OpenMP* STREAM benchmark. A future release of the PGI compilers and tools will include parallelization capabilities that *guarantee* alignment of subsections of arrays in parallel loops provided that the first element of the array is properly aligned.

- *–Mvect=nosizelimit* – generate vector code for all loops where possible regardless of the number of statements in the loop. This overrides a heuristic in the vectorizer that ordinarily prevents vectorization of loops with a number of statements that exceeds a certain threshold.

- *–tp { k7 | k8-32 | k8-64 | piii | p5 | p6 | p7 | p7-64 | px }* – Set the target architecture. By default, the *PGI CDK* compilers produce code specifically targeted to the type of processor on which the compilation is performed. In particular, the default is to use all supported instructions wherever possible when compiling on a given system. As a result, executables created on a given system may not be useable on previous generation systems (for example, executables created on a Pentium 4 may fail to execute on a Pentium III or Pentium II). Processor-specific optimizations can be specified or limited explicitly by using the *–tp* option. In this way, it is possible to create executables that are useable on previous generation systems. With the exception of *k8-64* and *p7-64*, any of these sub-options are valid on any x86 or AMD64 compatible system. The *k8-64* and *p7-64* sub-options are valid only on AMD Athlon64/Opteron processor-based systems, and processors such as the Intel Xeon EM64T which are designed to be AMD64 binary compatible, running a 64-bit operating system. Following is a list of possible sub-options to *–tp*, and the processors they are intended to target:

k7	generate 32-bit code optimized for AMD AthlonXP and compatible processors.
k8-32	generate 32-bit code optimized for AMD64 technology processors.
k8-64	generate 64-bit code optimized for AMD64 technology processors.
piii	generate 32-bit code optimized for Pentium III processors.
px	generate 32-bit code that is useable on any x86 processor.
p5	generate 32-bit code optimized for Pentium processors.
p6	generate 32-bit code optimized for Pentium Pro/II and processors.
p7	generate 32-bit code optimized for Pentium 4 and Xeon processors.
p7-64	generate 64-bit code optimized for Xeon EM64T processors.

4.5 64-bit Support

The *PGF77* and *PGF90* compilers included in *PGI CDK 5.2* support both the *–mcmodel=small* and *–mcmodel=medium* addressing models as defined in the *X86-64 Application Binary Interface*. Here are some terms useful to understand the capabilities of these programming models.

Address Type (**A**) – the size in bits of data used for address calculations, 32-bit or 64-bit.

Index Arithmetic (**I**) – the size in bits of data used to index into arrays and

other aggregate data structures. If **I** is 32-bit, the total range or size of any single data object is limited to 2GB.

Maximum Array Size (**AS**) – the maximum size in bytes of any single data object.

Maximum Data Size (**DS**) – the maximum size in bytes of the aggregate of all data objects in *.bss* sections in an executable.

Maximum Total Size (**TS**) – the maximum size in bytes, in aggregate, of all executable code and data objects in a running program.

The table below describes 32-bit versus 64-bit capabilities of *linux86* and *linux86-64* executables when programs are compiled and linked using various combinations of options to the *PGI CDK 5.2* compilers. The program area is the total area used by the Linux operating system and the user program. On most 32-bit Linux systems, only about 1GB is available for data (in theory 2GB is accessible with a 32-bit signed integer address).

Programming Models on 64-bit Linux86-64 Systems						
Combined Options to PGI Compilers	**Address Arithmetic**		**Maximum Data Size in Gbytes**			**Comments**
	A	**I**	**AS**	**DS**	**TS**	
–tp {k8-32 | p7}	32	32	2	2	2	Compatible with 32-bit *linux86* programs
–tp {k8-64 | p7-64}	64	32	2	2	2	64-bit addressing, but *–mcmodel=small* default limits data area
–tp {k8-64 | p7-64} –fpic	64	32	2	2	2	*–fpic* can't be used with *–mcmodel=medium* on compile line, but *–fpic* shared libs can be linked into either small or medium memory executables
–tp {k8-64 | p7-64} –mcmodel=medium	64	32	2	>2	>2	64-bit data area, but 2GB size limit on each data object, and a potential performance penalty since `%rip`-relative addressing can't be used
–tp {k8-64 | p7-64} –mcmodel=medium –Mlarge_arrays	64	64	>2	>2	>2	Used with *PGF77* and *PGF90* to enable full support for 64-bit data addressing

The (default) *small memory model* of the *linux86-64* environment limits the combined area for a user's object or executable to 1GB, with the Linux kernel managing usage of the other 1GB of address for system routines, shared libraries, stacks, etc. Programs are started at a fixed address, and the program can use a single instruction to make most memory references.

Support for the *medium memory model* in the *linux86-64* environment is provided using the *–mcmodel=medium* compile and link option. The

medium memory model allows for larger than 2GB data objects and *.bss* sections. Object files linked into an executable requiring the *–mcmodel=medium* link-time option must be compiled using either *–mcmodel=medium* or *–fpic*, but cannot be compiled using both of these options.

IMPORTANT NOTE: while *medium memory model* executables can incorporate both *–mcmodel=medium* objects and *–fpic* objects, it is important to reiterate that these two options *cannot* be used together on a given file. In particular, this means that it is not possible to create shared object libraries that include objects compiled *–mcmodel=medium*. This is a limitation of the *X86-64 Application Binary Interface*, not a limitation specific to the PGI compilers and tools.

The *linux86-64* environment provides system libraries in two forms, and the PGI compilers runtime libraries are provided in these same two forms:

- Static *libxxx.a* archives built without *–mcmodel=medium* and without *–fpic* (static *–mcmodel=small* archives)

- Dynamic *libxxx.so* shared object libraries that are compiled *–fpic*
  (dynamic *–mcmodel=small* archives)

The *–mcmodel=medium* linker switch implies the *–fpic* switch and will utilize the shared object libraries by default. ***NOTE***: a side-effect of this aspect of the *linux86-64* environment is that it is not possible to create statically-linked *–mcmodel=medium* executables. However, it is possible to create your own static archives built using *–mcmodel=medium*, and statically link objects from such archives into a *–mcmodel=medium* executable.

4.6.2 Practical Limitations of –mcmodel=medium

The 64-bit addressing capability of the *linux86-64* environment can cause unexpected issues when data sizes are enlarged significantly. For example:

- Initializing a large array with a data statement may result in very

large assembly and object files, where a line of assembler source is required for each element in the initalized array. Compilation and linking will be very time consuming as well. To avoid this issue, consider initializing large arrays in the program area in a loop rather than in the declaration.

- Stack space can be a problem for data that is stack-based. Issuing the command `limit stacksize unlimited` in your shell environment can enable as much stack space as possible, but it will be limited nonetheless and is dependent on the amount of physical memory. Determine if `limit stacksize 512M` gives as large a stack area as `unlimited`. If so, there is a hard limit to the stack size imposed by the operating system and the programmer must work around this if necessary by modifying the program to reduce the amount of data that is stack-based.

- If your executable is much larger than the physical size of memory, page swapping can cause it to run dramatically slower and it may even fail. This is not a compiler problem. Try smaller data sets to determine if a problem is due to page thrashing, or not.

- Be sure your *linux86-64* system is configured with swap space sufficiently large to support the data sets used in your application(s). If your memory+swap space is not sufficiently large, your application will likely encounter a segmentation fault at runtime.

Overall, it is important to understand the practical limitations of the *linux86-64* environment, and programmers should take reasonable care to determine if a program failure is due a compiler limitation or an operating system limitation.

4.6.2 Compiler Limitations of –mcmodel=medium

For the *PGHPF* and *PGC++* compilers included in PGI CDK 5.2, single data objects are still limited to less than 2GB in size. This limitation will be removed in a future release of the PGI compilers and tools.

4.6.2 Large Array Example in C

Consider the following example, where the aggregate size of the arrays exceeds 2GB.

```
% cat bigadd.c

#include <stdio.h>

#define SIZE 600000000  /*  > 2GB/4 */

static float a[SIZE],b[SIZE];
main() {
long long I,n,m;
float c[SIZE];  /* goes on stack */
n=SIZE;m=0;

   for(i=0;i<n;i+=10000){
      a[i]=i+1;
      b[i]=2.0*(i+1);
      c[i]=a[i]+b[i];
      m=I;
   }
   printf("a[0]=%g b[0]=%g c[0]=%g\n", a[0], b[0],
          c[0]);
   printf("n=%d a[%d]=%g b[%d]=%g c[%d]= %g\n", n, m, m,
          m, a[m], b[m], c[m]);
}
```

Compiled using *gcc*, without using *–mcmodel=medium*:

```
% gcc -o bigadd bigadd.c
/tmp/ccWt7q8Q.o: In function `main':
/tmp/ccWt7q8Q.o(.text+0x6e): relocation truncated to
fit: R_X86_64_32S .bss
/tmp/ccWt7q8Q.o(.text+0x8c): relocation truncated to
fit: R_X86_64_32S .bss
```

This is a link-time error, and is due to the linker attempting to create a *small memory model* executable when the static arrays exceed the aggregate limit inherent in that model. Re-compiling using *–mcmodel=medium*:

```
% gcc -mcmodel=medium -o bigadd bigadd.c
```

```
/tmp/ccVQpbPj.s: Assembler messages:
/tmp/ccVQpbPj.s:97: Error: .COMMon length (-2147483648.)
<0! Ignored.
```

The *gcc* compiler incorrectly converts a greater than 2G value to a *negative* **32**-bit number in an assembler statement. This error does not occur using *pgcc 5.2*:

```
% pgcc -mcmodel=medium -o bigadd bigadd.c
```

Why? When `SIZE` is greater than 2G/4, and the arrays are of type `float` with 4 bytes per element, the size of each array is *greater* than 2GB. With 5.2 pgcc, using the *–mcmodel=medium* switch, a static data object *can now be* > 2GB in size. Note that if you execute with the above settings in your environment, you may see the following:

```
% bigadd
Segmentation fault
```

Execution fails because the stack size is not large enough. Try resetting the stack size in your environment:

```
% limit stacksize 3000M
```

Note that '*limit stacksize unlimited*' will probably not provide as large a stack as we are using above.

```
% bigadd
a[0]=1   b[0]=2 c[0]=3
n=600000000 a[599990000]=5.9999e+08
b[599990000]=1.19998e+09 c[599990000]=1.79997e+09
```

The size of the *bss* section of the *bigadd* executable is now larger than 2GB:

```
% size -format=sysv bigadd | grep bss
.bss              4800000008   5245696
% size -format=sysv bigadd | grep Total
Total             4800005080
```

4.6.2 Large Array Example in Fortran

The following example works with both the *PGF90* and *PGF77* compilers included in *PGI CDK 5.2*. Both compilers use 64-bit addresses when the *–mcmodel=medium* option is used and both allow for 64-bit addressing and 64-bit integer index support if *–Mlarge_arrays* is also used.

Consider the following example:

```
% cat matadd.f
      program matadd
      integer I, j, k, size, l, m, n
      parameter (size=16000) ! >2GB
      parameter (m=size,n=size)
      real*8 a(m,n),b(m,n),c(m,n),d

      do I = 1, m
         do j = 1, n
            a(I,j)=10000.0D0*dble(i)+dble(j)
            b(I,j)=20000.0D0*dble(i)+dble(j)
         enddo
      enddo
!$omp parallel
!$omp do
      do I = 1, m
         do j = 1, n
            c(I,j) = a(I,j) + b(I,j)
         enddo
      enddo
!$omp do
      do i=1,m
         do j = 1, n
            d = 30000.0D0*dble(i)+dble(j)+dble(j)
            if(d .ne. c(I,j)) then
               print *,"err i=",I,"j=",j
               print *,"c(I,j)=",c(I,j)
               print *,"d=",d
               stop
            endif
         enddo
      enddo
!$omp end parallel
```

```
      print *, "M =",M,", N =",N
      print *, "c(M,N) = ", c(m,n)
      end
```

When compiled with the *PGF90* compiler using *–mcmodel=medium* and *–Mlarge_arrays*:

```
% pgf90 -mp -o matadd matadd.f -mcmodel=medium -Mlarge_arrays

% setenv OMP_NUM_THREADS 2
% matadd
M =        16000 , N =        16000
c(M,N) =     480032000.0000000
```

On a 1.8 GHz Dual processor Opteron box with 4GB of memory, the above example executes about 33% faster with OMP_NUM_THREADS set to 2, instead of 1.

4.7 PGDBG and PGPROF

PGI CDK 5.2 includes several new features in the *PGDBG* parallel debugger and *PGPROF* performance profiling tools. In particular, both of these tools include completely new graphical user interfaces (GUIs).

PGDBG is supported as a graphical and command line debugger in both the *linux86* and *linux86-64* execution and development environments. Like the compilers, *PGDBG* for *linux86-64* must run in a *linux86-64* execution environment. *PGDBG* for *linux86* environments is a separate version, and it will also run in the *linux86-64* execution environment, but only with *linux86* executables. The *linux86-64* version of *PGDBG* will only debug executables built to run as *linux86-64* executables. *PGDBG* for *linux86-64* has been enhanced to disassemble the new *AMD64* technology instructions and associated registers, and is more compatible with *gcc*, *g77*, and *g++* debug information.

PGPROF is supported as a graphical and command line profiler in both the *linux86* and *linux86-6* environments. The same version works in either the *linux86 or linux86-64* environment to process a trace file of profile data created by executing the instrumented program. Program instrumentation

is either line-level (*–Mprof=lines*), function-level (*–Mprof=func*), or *gprof*-style (*–pg*) sample based and trace profiling. Note: sample based and trace profiling is currently supported on single process execution only.

The new *PGDBG* and *PGPROF* graphical user interfaces (GUIs) are invoked by default. To use the command-line interfaces, invoke either tool with the *–text* option. To use the old GUI interfaces (included in *PGI CDK 5.1* and prior releases), invoke either tool with the *–motif* option.

4.6.2 PGDBG and PGPROF New Features

Following are the new features included in the *PGI CDK 5.2* versions of *PGDBG* and *PGPROF*:

- *Fortran 95 support – PGDBG* and *PGPROF* both support the language, syntax, and context of *Fortran 95*.

- *New Graphical User Interfaces (GUIs)* – all-new graphical user interfaces provide easier, more intuitive and effective ways to access the debugger and functionality. The ***PGDBG*** graphical interface supports single-threaded, multi-threaded, and distributed applications. The *PGPROF* graphical user interface supports either PGI-style `pgprof.out` trace files or *gprof*-style `gmon.out` trace files, including source correlation for *gprof*-style traces.

- *Process attach – PGDBG* now supports the *attach* and *detach* commands to attach and detach the debugger to or from running processes. This functionality works for MPI applications, allowing attach to all processes in the MPI application with a single attach command.

- *AMD64* `call` *command – PGDBG* now supports the `call` command for *linux86-64* environments, with some minor limitations (see below) in passing *F90* deferred shape array arguments.

- *Large Arrays – PGDBG* now supports *linux86-64* applications built with *–mcmodel=medium –Mlarge_arrays*.

- *NPTL threads support* – *PGDBG* now supports debugging of SMP parallel programs that use the *NPTL* threads package included in newer distributions of Linux.

- *Dynamic threads support* – In previous releases, *PGDBG* was unable to debug multi-threaded parallel programs built on some Linux distributions unless the programs were statically linked. *PGDBG* can now debug such programs even if they are dynamically linked.

- *gprof-style profiling* – Sample-based profiling, and the ability to read and display `gmon.out`-style trace files, is now supported in *PGPROF*.

- *Scalability comparisons* – PGPROF includes improved capability for scalability comparisons of multiple runs of a parallel program on different numbers of threads or processors.

- *Online help* – both *PGPROF* and *PGDBG* now have extensive online help facilities as part of the new GUIs. Most information available on individual debugger or profiler commands from the *PGI Tools Guide* is incorporated into these online help facilities.

See the *PGI Tools Guide*, completely updated for *PGI CDK 5.2*, for a complete description of the usage and capabilities of *PGDBG* and *PGPROF*. For tool limitations and workarounds, see the FAQ *http://www.pgroup.com/support/tools.htm*.

4.6.2 PGDBG and PGPROF Corrections

Following is a list of the technical problem reports (TPRs) filed in previous releases for PGDBG and PGPROF, and which have been corrected in *PGI CDK 5.2*:

PGDBG and PGPROF TPRs Corrected in PGI CDK 5.2-2				
TPR	**Rel**	**Lang/ tool**	**Description**	**Symptom**
2093	3.1	PGDBG	Fix display of `pgf90` pointer variables	`Display not right`
2315	3.1	PGDBG	*C++* language inconsistency	`Display not right`
2773	5.1	PGDBG	1. Printing allocatable arrays. 2. Source pane not updated after reload	`Display not right`
2806	5.1	PGDBG	print of F90 array pointer that is a field of a derived type.	`prints only the first element`
3224	5.1	PGDBG	Documentation corrected to say core files are **NOT** supported.	`Documentation said core files supported`
3178	5.1	PGF90	`pgf90` producing faulty debug information	`pgdbg failed`

4.7 Known Limitations

The frequently asked questions (FAQ) section of the *PGI* web page at *http://www.pgroup.com/support/faq.htm* provides more up to date information about the state of the current release.

- While object files created using *PGI CDK 5.2* compilers are compatible with object files from previous releases, module files are not. Fortran 90 program units which include or use modules must be re-compiled in order to be successfully used in an executable created using *PGI CDK 5.2* compilers.

- Programs that incorporate object files compiled using *–mcmodel=medium* cannot be statically linked. This is a limitation of the *linux86-64* environment, not a limitation specific to the PGI compilers and tools.

- Using *–Mipa=vestigial* with *PGCC*, you may encounter unresolved references at link time. This is due to the erroneous removal of functions by the *vestigial* sub-option to *–Mipa*. You can work around this problem by listing specific sub-options to *–Mipa*, not including *vestigial*

- Using *–Mprof=func*, *–mcmodel=medium* and *–mp* together on any of the PGI compilers can result in segmentation faults by the generated executable. These options should not be used together.

- Programs compiled and linked for *gprof*-style performance profiling using *–pg* can result in segmentation faults on system running version 2.6.4 Linux kernels. In addition, the time reported for each program unit by *gprof* and *PGPROF* for such executables run under some Linux distributions can be a factor of 10 higher than the actual time used. This is due to a bug in certain shared object libraries included with those Linux distributions.

- *OpenMP* programs compiled using *–mp* and run on multiple processors of a *SuSE 9.0* system can run very slowly. These same executables deliver the expected performance and speed-up on similar hardware running *SuSE 9.1*. This problem is still being diagnosed, and will be fully documented and corrected if possible in a future release of the PGI compilers.

- *PGDBG* cannot print the values of *PRIVATE* variables while debugging Fortran threads in an *OpenMP* parallel region.

- *PGDBG* now supports the source-level debugging of shared objects, but a shared object must be loaded before it can be debugged using *PGDBG*.

- When linking the *ACML* using the *–lacml* option, you must also include the *–mp* option at link time. The *ACML* is only provided in an *OpenMP*-capable version at this time. A true serial version of the *ACML* will be included in a future release of the PGI compilers and tools.

- Attaching to a running process using the menu selection "File->Attach to Debugger..." from the *PGDBG* GUI may produce spurious error messages in the command prompt panel and/or the program I/O Window. These messages should be ignored.

- *PGPROF* only supports viewing of *gprof*-style trace files under

the new GUI. This capability is not supported by the old *motif* GUI or the command-line version of *PGPROF*.

- Times reported for multi-threaded sample-based profiles (*gprof*-style profiles) are cumulative. PGI-style instrumentation profiling with *–Mprof={lines | func}* must be used to obtain profile data on individual threads or processes.

Previous releases of the *PGI CDK* Linux compiler products have included a customized version of *libpthread.so* called *libpgthread.so*. The purpose of this library is to give the user more thread stack space to run *OpenMP* and *–Mconcur* compiled programs. With Release 8.0 Red Hat and equivalent releases, *libpthread.so* and *libpthread.a* have 're-sizeable' thread stack areas. In these cases

- The filename *$PGI/linux86/5.2/lib/libpgthread.so* is a soft link to */usr/lib/libpthread.so*.

- Instead of 'setenv MPSTKZ 256M', for example to increase the *libpgthread.so* thread stack area, the Linux system call 'limit stacksize 256M' now applies to thread stacks.

On *linux86-64* systems, the 32-bit Linux *libpthread* libraries appear to no longer have floating stacks, but actually reset the stack size to 2MB if linked into an executable. The Portland Group Compiler Technology considers this a bug. This behavior is not exhibited by 64-bit *libpthread* implementations. This behavior has only been observed for 32-bit *libpthread* libraries included in *linux86-64* environments.

4.8 Corrections

The following problems have been corrected in the *PGI CDK 5.2* release. Most were reported in *PGI CDK 5.1-6* or previous releases. Problems found in *PGI CDK 5.1-6* may not have occurred in the previous releases. A table is provided that describes the summary description of the problem. An *Internal Compiler Error* (ICE) is usually the result of checks the compiler components make on internal data structures, discovering

inconsistencies that could lead to faulty code generation.

Compiler Technical Problem Reports (TPRs) Corrected in PGI CDK 5.2-2				
TPR	**Rel**	**Lang/ tool**	**Description**	**Symptom**
2800	3.3	pghpf	-g with automatic arrays	Signal 11
2811	4.0	pgf90	Program makes compiler fail	ICE
2863	4.0	pgf90	Incorrect severe error	Severe error
2864	4.0	pgf90	Incorrect severe error	Severe error
2876	4.0	pgf77	Program makes compiler fail	ICE
2879	4.0	pgf90	Defective program does not cause error	No errors
2899	4.0	pgf90	Fails with –i8	ICE
2911	4.0	pgf90	False error	Severe error
3005	5.0	pgf90	-mcmodel=medium	Bad answers
3017	5.0	pgf90	False error	Severe error
3022	5.0	pgf77 pgf90	Large array example	Bad answers
3029	5.0	pgf90	False errors	errors
3033	5.0	pgCC	Bad behavior with optimization	Work –O0, not –O1
3034	5.0	pgCC	Program fails on linux86-64	Seg fault
3039	5.0	pgf77	Pgf77 fails -fast	ICE
3044	5.0	pgf90	Problems with driver	X86 vs x86-64
3059	5.0	pgf90	Program breaks compiler	ICE
3069	5.0	pgf90	Compiler breaks with program	Signal 11
3073	5.0	libpgmp	Program fails because of 32-bit libpthread	Linux86-64 limit
3075	5.0	pgf90	Program gives bad answers	Strings false
3096	5.0	pgCC	Compiler breaks with error C++	ICE
3097	5.0	pgf90	Argument mismatch errors	errors
3102	5.1	pgf90	Random works inconsistently	Different answers
3105	5.1	pgCC	-mp works wrong with inlining	Different answers
3131	5.1	pgf90	Program causes false severe errors	Severe errors
3145	5.1	pgf90	Invalid pointer problem	False error
3162	5.1	pgf90	Different answers with -)2 -Munroll	Different answers
3163	5.1	pgf90	Assumed shape error	Bad answers
3168	5.1	pgf90	Program breaks compiler	Signal 11
3170	5.1	pgf90	Equivalence problem	Bad answers
3193	5.1	pgf90	-Mbounds false error	Bounds errors
3198	5.1	pgf90	Bad answers with –O2	Bad answers
3205	5.1	pgf90	Program generates false pgf90 errors	Severe errors
3209	5.1	pgf90	Program breaks compiler	ICE
3210	5.1	pgf90	Program breaks compiler	ICE
3211	5.1	pgf90	Program breaks compiler	ICE
3216	5.1	pgf90	Internal procedure error	Bad answers

3217	5.1	pgf90	Bad pointer assignment	Bad answers
3218	5.1	pgf90	Internal procedure error	Bad answers
3219	5.1	pgf90	Alignment errors	Bad answers
3220	5.1	pgf90	Program gives wrong results	Bad answers
3221	5.1	pgf90	Regression in behavior over 5.0	Bad answers
3222	5.1	pgf90	Test routine gives bad answers	Test Fails
3223	5.1	pgf90	Test routine gives bad answers	Test Fails
3224	5.1	pgf90	Test routine fails	Test Fails
3225	5.1	pgf90	Different answers from5.0	Bad answers
3226	5.1	pgf90	Test routine gives bad answers	Test Fails
3227	5.1	pgf90	Test routine gives bad answers	Test Fails
3230	5.1	pgf90	Program causes pgf90 false errors	Severe errors
3231	5.1	pgf90	Internal procedure error	Bad answers
3232	5.1	pgf90	Internal procedure error	Bad answers
3233	5.1	pgf90	Internal procedure error	Bad answers
3234	5.1	pgf90	Program produces false answers	Bad answers
3238	5.1	pgf90	Program crashes	Seg fault
3243	5.1	pgf90	Executable has PGDBG_stub out of range	R X86 64 32
3271	5.1	pgf90	-Msave causes compiler to break	ICE
3274	5.1	pgC++	Lam-mpi was not building	Compile fails
3276	5.1	Pgf90	False errors reported	Compile errors
3277	5.1	Pgf90	Wrong answers	Bad answers
3287	5.1	Pgf90	WRF2 fails to compile	ICE - dt nfd
3302	5.1	driver	System libraries linked in wrong order	Bad linking
3303	5.1	Pgf90	Severe error for correct code	Type mismatch
3311	5.1	Pgf90	-O2 causes a signal 11 termination	Signal 11

5 PGHPF 5.2

PGHPF 5.2 is PGI's native (HPF-to-assembly code) High Performance Fortran compiler for *linux86* and *linux86-64* environments. All features of Full HPF 1.1 and Fortran 90 are supported, with the few exceptions noted in this document. In addition, several new HPF 2.0 and HPF/JA features have been added in this release. Section 4.1 below contains a list of features, which is a continuation of the 5.2 release information. If you encounter any feature that is not supported, and not listed in section 4.3, *Restrictions*, please consider it a bug and report it to PGI at the e-mail address *trs@pgroup.com*.

5.1 Summary of Changes

The following features have been added to *PGHPF 5.2* for IA-32 systems:

- HPF/JA feature - support for `REDUCTION` type in `INDEPENDENT` clauses, including the new reduction operators `FIRSTMAX`, `LASTMAX`, `FIRSTMIN`, and `LASTMIN` and relaxation of the restrictions on allowable forms of references to reduction variables when the reduction type is specified.

- HPF 2.0 feature – support for the approved extension form of the `ON` directive, restricted to the body of an `INDEPENDENT` loop

- HPF 2.0 feature – the HPF library procedures `SORT_UP` and `SORT_DOWN` are now supported
- Reductions in nested `INDEPENDENT` loops are now supported
- Constraints on the order of HPF mapping directives have been eliminated
- Scaling analysis of HPF programs using PGPROF 5.2 – see section 3.12.1 for more information on this feature

5.2 Restrictions

This section lists Fortran 90 and HPF features that are *not* supported in *PGHPF 5.2*.

Type	Restriction

Fortran 90 Pointers	• Objects with the `POINTER` attribute cannot be `DYNAMIC`
	• Objects with the `TARGET` attribute cannot have `CYCLIC` or `CYCLIC(N)` distributions. It may not be possible to detect this at compile-time in all cases, for example when a `CYCLIC` actual argument is passed to a dummy with the `TARGET` attribute
	• A scalar `POINTER` cannot be associated with a distributed array element. For example
	`integer, pointer :: p`
	`integer, target :: a(10),b(10)`
	`!hpf$ distribute (block) :: a`
	`p=> a(1) ! unsupported`
	`p => b(1) ! supported`
	`end`

	• A `POINTER` dummy variable cannot be used to declare other variables such as automatic arrays using the `lbound()`, `ubound()` and `size()` intrinsics. For example:
	
	`subroutine sub(p)`
	`integer, pointer, dimensions(:,:) :: p`
	`integer, dimension(lbound(p,1): & +ubound(p,1),size(p,2)) :: a ! does not work`

Fortran 90 Derived Types	The `DATA` statement does not work with arrays of derived type. As a work-around, use entity-style initialization.

Named Constants	Named multi-dimensional array constants cannot be subscripted to yield a constant value. For example, given the declaration:
	
	`INTEGER, PARAMETER, DIMENSION(2,2) :: & X = RESHAPE((/1,2,3,4/),(/2,2/))`
	
	the following will not work:
	
	`INTEGER, PARAMETER :: Y = X(1,2) ! Will not work`
	
	Because of this restriction, named multi-dimensional array constants cannot be used in:
	
	• Values in `CASE` statements
	• `KIND` parameters in declaration statements
	• `KIND` arguments to intrinsics

	Initial values in parameter statements or type declaration statements
HPF Library	The `HPF_LIBRARY` routines `GRADE_UP`, `GRADE_DOWN`, `SORT_UP` and `SORT_DOWN` require a `DIM` argument. These routines do not support cyclic distributions of the selected dimension.
PURE Procedures	The *PGHPF 5.2* implementation of `PURE` conforms to the HPF 2.0 language specification, with the following exception: in `PURE` subroutines *PGHPF* will not generate any communication for distributed `COMMON` variables or distributed `MODULE` variables. The user is advised to pass distributed `COMMON` variables as arguments to a `PURE` subroutine, or use non-distributed `COMMON` variables.
Optional Arguments	An F90 optional argument cannot be used as an *align-target* for any variable that is not also optional. If an *alignee* is present, then the *align-target* must also be present.
DISTRIBUTE and ALIGN	The following compile-time warning message: `PGHPF-W-301 – Non-replicated mapping for character/struct/union array, char_table, ignored (file.F: lineno)` indicates that *PGHPF 5.2* ignores the distribution directives applied to character arrays, arrays subject to `SEQUENCE` directives, and `NAMELIST` arrays.
INDEPENDENT loops	At present, only `INDEPENDENT` loops containing FORTRAN 77 constructs can be parallelized. In particular, the presence of array assignments, `WHERE` statements, `FORALL` statements, and `ALLOCATE` statements will eliminate `INDEPENDENT` loops from consideration for parallelization.

6 Contacting PGI & Online Documentation

You can contact us at the following address:

The Portland Group Compiler Technology
STMicroelectronics,Inc.
9150 SW Pioneer Court
Suite H
Wilsonville, OR 97070

Or contact us electronically using any of the following means:

Fax: *+1-503-682-2637*
Sales: *sales@pgroup.com*
Support: *trs@pgroup.com*
WWW: *http://www.pgroup.com*

Online documentation is available by pointing your browser at either your local copy of the documentation:

```
file:$PGI/doc/index.htm
```

or at our Web site:

```
http://www.pgroup.com
```