PGI® CDK™ 5.1
Cluster Development Kit®

Installation & Release Notes 5.1-3

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

PGI 5.1 Cluster Development Kit (CDK™)
Installation & Release Notes

First Printing: Release 5.1, December 2003

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

Table of Contents

1 PGI CDK 5.1 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 5.1,* the 64-bit AMD Opteron (AMD64 technology) processor-based systems is 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 5.1 is the first production release of *PGI CDK* that supports AMD64 technology processor-based systems, with large array addressing in pgf77 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. Executables generated by 32-bit x86 compilers and tools, such as previous releases of the *PGCC*, *PGC++*, *PGF77*, or *PGF90* compilers from The Portland Group Compiler Technology, or the open source *gcc* compiler for

32-bit Linux operating systems, can execute unchanged on **AMD64** technology processor-based systems. Applications that are re-compiled to take advantage of the new features of **AMD64** technology processor-based systems can realize significant performance improvements.

The *PGI CDK Release 5.1* compilers have been enhanced to improve performance of generated executables through improved global optimization, vectorization and inter-procedural analysis (IPA). In addition, scheduling and code generation optimizations have been improved on 32-bit x86 targets, and a completely new code generator is provided to optimize for **AMD64** technology targets running 64-bit Linux. Improved performance is reflected in executables generated for either 32-bit x86 processor-based systems or AMD64 technology processor based systems.

The *PGI CDK Release 5.1* products can be installed in two types of code development environments:

1. **linux86** – 32-bit x86 processor-based Linux systems, with 32-bit GNU tools, utilities and libraries used by the *PGI CDK* compilers to assemble and link for execution.

2. **linux86-64** – 64-bit **AMD64** technology processor-based systems running 64-bit **SuSE Linux Enterprise Server 8**, version 8.1, including both 64-bit and 32-bit GNU tools, utilities and libraries used by the *PGI CDK* compilers to assemble and link for execution. In this environment, the *PGI CDK Release 5.1* compilers can produce either **AMD64** technology or legacy 32-bit x86 Linux executables. The 32-bit development tools and execution environment under linux86-64 are considered a cross development environment for x86 processor based applications.

For purposes of this document, we will refer to applications compiled to use the AMD64 technology 64-bit addressing abilities as *linux86-64* executables, and programs compiled to run on Linux systems with 32-bit x86 processors as *linux86* executables. It is important to remember that in a *linux86-64* environment, both *linux86-64* executables *and linux86* executables will run successfully. However, it is not the case that *linux86-64* executables will run in a *linux86* environment. Note that the compilers

that create *linux86-64* executables **are** *linux86-64* executables themselves, so it is not possible to install those compilers on any platform other than the one described in **2** above.

The 5.1-3 release of the CDK marks the first support of the *linux86-64* medium memory model that extends the individual data object size in pgf77 or pgcc to beyond 2GB. Data objects of significantly greater size can be handled within the constraints of the linux86-64 system's configuration (for example the amount physical memory resident on your machine can limit the size of programs you can run).

It is anticipated that many users will want to build and run programs from identical source code as either *linux86* executables or as *linux86-64* executables, except for the potential size of data structures. It is key that these programs be ported to run as *linux86* executables prior to building them as *linux86-64* executables, to ensure no 64-bit dependencies exist. As a methodology for developing such applications, users can build programs with *PGI CDK Release 5.1* targeting 32-bit *linux86* environments on an AMD64 technology processor-based system running 64-bit Linux (cross-compilation). These same applications can then be recompiled on an AMD64 technology processor-based system running 64-bit Linux without source code changes to target *linux86-64* environments (native compilation).

In general, developers should find migration to the *linux86-64* environment much easier when they can compile, profile, and debug a *linux86* executable in one window while also compiling, profiling, and debugging the same program, built as a *linux86-64* executable, in a different window. In fact, it is possible to incrementally migrate portions of an application to the *linux86-64* environment if the application comprises multiple executables. For example, it is possible for a *linux86* executable and a *linux86-64* executable running in a *linux86-64* environment to communicate using sockets.

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.

1.1 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 1.2 or contact your system administrator.

Release 5.1 of the *PGI CDK* consists of the following PGI compilers and tools, that will develop *linux86* and *linux86-64* executables:

- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86*, and *linux86-64* environments.
- *PGF90* native OpenMP and auto-parallelizing Fortran 90 compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGF77* native OpenMP and auto-parallelizing F77 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.
- The Multi-process, Multi-thread supporting *PGPROF* graphical profiler that will run on *linux86* and *linux86-64*.

- The Multi-process, Multi-thread supporting *PGDBG* graphical debugger in versions that will run on linux86 and *linux86-64* development environments.

the following open source clustering utilities:

- *MPI-CH* version 1.2.5, an implementation of the Message-Passing Interface (MPI) standard, compiled for use with 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.
- *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 *MPI-CH* 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.
- *PBS* portable batch queuing system from *Veridian Technologies*, version 2.3.15, configured for Linux systems with a kernel revision of 2.2.10 or higher. This is installed only as a *linux86* process, though *linux86-64* executables can be handled.

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.

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 *MPI-CH* in the first configuration, *PBS* 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 *MPI-CH* 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 *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 MPI-CH and 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 PBS, which users at your site will have 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 1.2 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 1.2 and then follow the instructions in section 1.3 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 2.1 - 2.5 describe basic usage of the Open Source components of the *PGI CDK*, including *MPI-CH*, the *PBS* batch queuing system, *ScaLAPACK* libraries, and the example benchmark programs and tutorials.

For the first 60 days after your purchase, you may send technical questions about the *PGI CDK* compilers and tools to the e-mail address *trs@pgroup.com*. If you have purchased PGI's Subscription Service, you will have access to e-mail 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.

MPI-CH, *PBS*, 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:

- *MPI-CH* - `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 *MPI-CH* developers.

- *PBS* - `http://www.openpbs.org` is the main site for the

PBS product we integrate with the CDK.

- *ScaLAPACK*- http://www.netlib.org/scalapack contains FAQ files and current distributions of *ScaLAPACK*.

Once you've completed installation of the *PGI CDK*, the MAILUSERS file on the *PGI CDK* CD-ROM includes a simple text template for an introductory e-mail to end-users of your cluster system. It explains how to initialize one's environment, access the man pages and documentation, and execute a simple program on the cluster.

1.2 Installing on Linux86 or Linux86-64

Those familiar with previous releases of PGI CDK should note that the installation directory structure has changed. The path to the PGI CDK 5.1 Release compilers must be modified accordingly.

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/linux86/5.1/bin	*linux86* versions of the compilers and tools
/usr/pgi/linux86/5.1/lib	*linux86* versions of the libraries, created on all platforms in the cluster.
/usr/pgi/linux86/5.1/liblf	*linux86*-only large-file-support *(-Mlfs*)

	versions of the libraries.
`/usr/pgi/linux86/5.1/include`	*linux86* versions of header files
`/usr/pgi/linux86-64/5.1/bin`	*linux86-64* versions of the compilers and tools
`/usr/pgi/linux86-64/5.1/lib`	*linux86-64* versions of the libraries, created on all platforms in the cluster. Not to be used for ***–mcmodel=medium*** development.
`/usr/pgi/linux86-64/5.1/libso`	*linux86-64* ***–fPIC*** libraries for ***–mcmodel=medium*** support
`/usr/pgi/linux86-64/5.1/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 where you have appropriate permissions. 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, *MPI-CH*, and *ScaLAPACK* in the $PGI directory tree in the appropriate bin, include, lib, and man subdirectories. It will also install *PBS* in the /usr/local/pbs and /var/spool/pbs directories, and start the *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 *MPI-CH*, *ScaLAPACK*, and the *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: 5.1-3
```

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/linux86/5.1/bin\
  /usr/local/pbs/bin $path )
```

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

```
% set path = ( usr/pgi/linux86-64/5.1/bin\
  /usr/local/pbs/bin $path )
```

and for the common documentation:

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

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

```
% export PATH=\
 /usr/pgi/linux86/5.1/bin:/usr/local/pbs/bin:$PATH
```

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

```
% export PATH=\
 /usr/pgi/linux86-64/5.1/bin:/usr/local/pbs/bin:$PATH
```

and for the common documentation:

```
% export MANPATH=\
$MANPATH:/usr/pgi/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 1.3) 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 1.3 to enable floating license capability, and the following sections for basic usage instructions for *MPI-CH*, *PBS*, and *ScaLAPACK*.

1.3 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 Globetrotter Software. 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 1.2 above.

Step 2 – Once you have obtained permanent license keys (see section 1.2 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>/linux86/5.1/bin/pgroupd
```

```
FEATURE pghpf-linux86 pgroupd 5.100 31-dec-0 1 \
2B9CF0F163159E4ABE32 VENDOR_STRING=123456:16:cdk \
HOSTID=<hostid> ck=49

FEATURE pghpf-linux86-64 pgroupd 5.100 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 5.1 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 **5.1** license. Contact ***trs@pgroup.com*** if you wish to have the **5.1** 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/linux86/5.1/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/linux86/5.1/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/linux86/5.1/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 *PBS* daemons on each node of your cluster. If you find that your default queue is not operating properly for some reason, try restarting *PBS* 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.

2 Using the Open Source Cluster Utilities

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

2.1 Running an MPI-CH 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 -lfmpich -lmpich
% 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 *PBS*, you should also try submitting a batch job to make sure your *PBS* default queue is operational. There is an example *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 PBS NOTE 1

*A batch job submitted using the PBS **qsub** command does not by default inherit the environment of the spawning shell. Instead, 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 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, PBS jobs will quietly fail in ways that are difficult to diagnose.*

IMPORTANT PBS NOTE 2

*In order for a 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 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, PBS jobs can fail to execute in ways that are difficult to diagnose.*

Now, try submitting a *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 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 *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 2.5, *Testing and Benchmarking*.

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

2.2 More About PBS

PBS (the Portable Batch-queuing System) is a very configurable batch scheduler developed by the NASA Ames Research Center and Veridian Technologies.

The `installcdk` script installs *PBS* 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 *PBS Administration Guide*, which is found in the `cdk/pbs` subdirectory of the *PGI CDK* CD-ROM.

Check that the *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 *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 *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.*

As mentioned in the introduction, *PBS* is very configurable. The above steps provide a simple means to install *PBS* and establish a single default space-shared queue. We strongly encourage your cluster administrator to print out the file `cdk/pbs/pbs_admin_guide.ps` and read through it to learn more about *PBS* and how it can best be used on your cluster. There is also a very active mail list for *PBS*, which you can learn more about by browsing the main *PBS* web page at `http://www.openpbs.org`.

2.3 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/linux86/5.1/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 *MPI-CH*. 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/linux86/5.1/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 2.4, *Testing and Benchmarking*.

2.4 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
% 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 *PBS* batch queuing system. There is a pre-configured *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 *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 5 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.

MPI-CH - These tests measure latency and bandwidth of your cluster interconnect for MPI-CH 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
```

3 PGI CDK 5.1 Release Notes

This document describes changes between *PGI CDK 5.1* 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/faq/new_rel.htm for the most recent information not included here.

3.1 Supported Systems and Licensing

PGI CDK Release 5.1 is supported on 32-bit Intel Pentium II/III/4/Xeon and AMD Athlon/AthlonXP (x86) processor-based systems**,** and 64-bit AMD Opteron (AMD64 technology) processor-based systems running:

- A 32-bit *linux86* environment with a kernel version of 2.2.10 or above, including versions of Linux that use *glibc2.2.x*, such as Red Hat 7.0 to 9.0, and SuSE 7.1 to 9.0.

- A 64-bit *linux86-64* environment with a kernel version of 2.4.19 or above, including versions of Linux that use *glibc2.2.5*, such as SuSE Linux Enterprise Server 8 (supported only on AMD64 technology processor-based systems), Suse 8.1, Suse 9.0, and Red Hat Enterprise Linux 3.0 .

For more information about release levels and operating systems supported, go to http://www.pgroup.com/faq/install.htm.

The *PGI CDK* Fortran, C, and C++ compilers are license managed. The open source components of the *PGI CDK*, including *MPI-CH*, *ScaLAPACK*, and *PBS*, are open source software packages that are not license-managed. For the *PGI CDK* compilers, the FLEXlm license manager controls the number of simultaneous users. When the *PGI CDK* compilers are first installed, they are usable for 15 days without a license key. Please contact PGI to obtain a permanent license key as soon as possible.

To make the *PGI CDK* compilers operational, you will need to follow the installation instructions in Section 1 above, including installation of the license daemon.

3.2 PGI CDK 5.1 Contents

Release 5.1 of the *PGI CDK* consists of the following PGI compilers and tools that will develop *linux86* and *linux86-64* executables:

- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86*, and *linux86-64* development environments.
- *PGF90* native OpenMP and auto-parallelizing Fortran 90 compiler, in versions that will run and produce code for execution in *linux86* and *linux86-64* development environments.
- *PGF77* native OpenMP and auto-parallelizing F77 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.
- The Multi-process, Multi-thread supporting *PGPROF* graphical

profiler that will run on *linux86* and *linux86-64*.

- The Multi-process, Multi-thread supporting *PGDBG* graphical debugger in versions that will run on linux86 and *linux86-64* development environments.

Also, the following open source clustering utilities are provided:

- *MPI-CH* version 1.2.5, an implementation of the Message-Passing Interface (MPI) standard, compiled for use with 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.
- *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 *MPI-CH* 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.
- *PBS* portable batch queuing system from *Veridian Technologies*, version 2.3.15, configured for Linux systems with a kernel revision of 2.2.10 or higher. This is installed only as a *linux86* process, though *linux86-64* executables can be handled.

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.

3.3 New Features for 32-bit x86 and AMD64

Following are the new features included in *PGI CDK 5.1*:

- A number of optimizations have been made to improve 32-bit x86 register allocation, vectorization, scalar arithmetic, loop unrolling, function inlining, scheduling, interprocedural analysis and optimization (IPA), and various other optimization phases within the compilers. SPECFP2K performance on 32-bit x86 processor targets is improved by 42% over the *PGI Workstation Release 4.1*, and SPECFP2K performance on AMD64 technology processor-based systems improves an additional 10% when re-compiled for *linux86-64* execution. Performance improvements in user-developed benchmarks or applications will of course vary.
- Support for vectorization of loops that operate on integer data
- Fortran 90 language and runtime optimizations
- Tuned Fortran 90 intrinsic MATMUL support libraries for all data types
- Additional interprocedural analysis and optimization (IPA) enhancements for Fortran 90, specifically including propagation of array shape information
- Improved interoperability with the Etnus TotalView debugger on *linux86* platforms.
- There are several enhancements to the *PGDBG* debugger:
  1. When the *–g* compiler option is specified, the compilers

emit debug information in DWARF2 format, improving interoperability with 3rd party debuggers, and better compatibility with *gcc* and *g77*.

2. *PGDBG* has made advances in internal information management, so that large programs on large systems will run more efficiently under *PGDBG* with faster load times and event handling.
3. Thread handling has improved, and *PGDBG* navigates threads with greater stability.
4. Source level debugging from shared libraries is now supported. A shared object must be loaded before it can be debugged using *PGDBG*.

- New installation directory structure. Users simply add */usr/pgi/linux86/5.1/bin* to their path, instead of */usr/pgi/linux86/bin*, when the install directory is */usr/pgi*. Users of the *linux86-64* versions of the *PGI Workstation* compilers add */usr/pgi/linux86-64/5.1/bin* to their path to use the AMD64 technology version of the compilers and tools by default. This change in directory structure enables installation and use of the *PGI Workstation* compilers for both *linux86* and *linux86-64* targets on AMD64 technology processor-based systems. It will also allow installation and use of multiple releases of the *PGI Workstation* compilers and tools for all releases subsequent to Release 5.0 (using the *–V* switch will tell you which release you are using for a given compilation). Installing the 5.1 release in /usr/pgi will not disturb the 5.0 release installation that was also installed in /usr/pgi.
- ACML high-performance LAPACK and BLAS math libraries bundled.
- Fully ISO-compliant *C++*, with the exception of exported templates.
- Completely updated hardcopy and online documentation.

- A new environment variable MP_WARN, to enable/disable runtime openmp warning messages.

3.4 New Features Exclusive to AMD64

Following are the new features included in *PGI CDK 5.1* that are specific to AMD64 technology processor-based systems.

- Further tuning of the AMD64 code generator, resulting in SPECFP2K Fortran benchmarks geometric mean improved by over 8%, Polyhedron geometric mean improved by over 25%, and significantly improved overall performance on the AMD Core Math Library (ACML) relative to *PGI Workstation 5.0-2*.
- Support for the 64-bit *linux86-64* execution environment and the x86-64 *medium memory model*.
- Support for 64-bit addressing and aggregate data sets in excess of the 2GB limit on 32-bit x86 processor-based. Single static data objects larger than 2GB are now supported in *PGF77* and *PGCC*.
- Support for 64-bit integer arithmetic in hardware.
- Use of the extended general-purpose register set available on AMD64 technology processors (16 general-purpose registers instead of the 8 available on 32-bit x86 processor-based systems).
- Use of the extended Streaming SIMD Extensions (SSE) register set available on AMD64 technology processors (16 SSE registers instead of the 8 available on 32-bit x86 processor-based systems).
- Most 32-bit and 64-bit floating-point arithmetic is performed using SSE instructions. *Note*: SSE instructions use standard IEEE 32-bit and 64-bit arithmetic for floating point calculations

and rounding behavior. The x87 floating-point stack used on x86 processor-based systems uses IEEE 80-bit arithmetic for register-to-register operations by default. As a result, arithmetic results on AMD64 technology processor-based systems running a *linux86-64* environment should closely match results obtained on most 32-bit and 64-bit RISC processor-based workstations. These results can sometimes differ from those obtained on a 32-bit x86 or AMD64 processor-based system running a *linux86* environment, where scalar arithmetic is by default performed using x87 instructions.

- Subroutine and function calling sequences are modified to comply with the *x86-64 Application Binary Interface*, which can be found online at http://www.x86-64.org/documentation.
- The *linux86-64* compilers themselves are 64-bit applications, and can only run on AMD64 technology processor-based machines.
- *PGDBG* supports debugging of *linux86-64* executables.
- *PGPROF* supports graphical display of *linux86-64* profiled output.
- In addition to SuSE Linux Enterprise Server 8 SP2, SuSE 9.0, and RHEL 3.0. are now also supported linux86-64 environments.
- Support for the 64-bit *linux86-64* execution environment

3.5 New Compiler Options

3.5.1 Getting Started

By default, the *PGI CDK* compilers generate code optimized for the type of processor on which compilation is performed (the compilation host). Typically, for best Fortran performance, 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. In a future release of *PGI CDK*, the *–fastsse* option will be deprecated in favor of a single *–fast* option which incorporates SSE/SSE2 optimizations. 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. See the *PGI User's Guide* for details on how to use this option. In particular, note that *–Mipa* requires two passes and that you must compile and link with *–Mipa* for it to be effective.

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.

For OpenMP programmers, runtime warnng messages can now be disabled via the **MP_WARN** environment variable. Setting it to *no* will prevent messages like "Warning OMP_NUM_THREADS greater than the number of cpus" from being posted.

3.5.2 New Linux Compiler Options

The following compiler options have been added in *PGI CDK Release 5.1*:

- *–Mlarge_arrays* – applies only to the PGF77 compiler. This option must be used, in combination with –mcmodel=medium, when compiling F77 applications that use single data objects larger than 2GB in size.

- *–Mdwarf1* | *–Mdwarf2* – generate debug information in either the DWARF1 or DWARF2 format. The Release 5.1 compilers produce DWARF2 by default. Must be used with *–g*.

- *–fast* – has been modified to be equivalent to: *–O2 –Munroll=c:1 –Mnoframe –Mlre*.

- *–fastsse* – has been modified to be equivalent to: *–fast –Mvect=sse –Mscalarsse –Mcache_align –Mflushz*.

- *–Minline*[={... | *except:<func>*}] – A new sub-option *except:<func>* directs the compiler to not inline an entry *func* or list of entries.

- *–tp* { *k7* | *k8-32* | *k8-64* | *piii* | *p5* | *p6* | *p7* | *px* } – Set the target architecture. By default, the *PGI Workstation* 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*, any of these sub-options are valid on any x86 or AMD64 technology processor-based system. The *k8-64* sub-option is valid only on AMD64 technology processor-based systems 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 for AMD AthlonXP and compatible processors.
k8-32	generate 32-bit code for AMD64 technology and compatible processors.
k8-64	generate 64-bit code for AMD64 technology and

compatible processors.

piii	generate 32-bit code for a Pentium III processor-based system.
px	generate 32-bit code that is useable on any x86 processor-based system.
p5	generate 32-bit code for Pentium and compatible processors.
p6	generate 32-bit code for Pentium Pro/II and compatible processors.
p7	generate 32-bit code for Pentium 4 and compatible processors.

- The environment flag **MP_WARN** controls the runtime posting of OpenMP warnings. '*setenv MP_WARN no'* will prevent OpenMP runtime warnings.

3.6 PGDBG and PGPROF Support

PGDBG is supported as a graphical 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* execution 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 AMD64 technology new instructions, and is more compatible with *gcc*, *g77*, and *g++* debug information.

PGPROF is a graphical display tool of profile information created through execution of programs compiled and linked using *–Mprof*. Only a *linux86* executable version of *PGPROF* is provided, but it is able to read either types of profile output, and resides in both *linux86* and *linux86-64 bin*

areas.

See the *PGI Tools Guide* for a complete description of the usage and capabilities of *PGDBG* and *PGPROF*.

3.7 Problems Corrected in Release 5.1

The following problems are corrected in the current release. A description of the problem is given, but some problems can only be described in general terms because of complexity or confidentiality.

Technical Problem Reports (TPRs) Corrected in PGI Release 5.1-3				
TPR	Rel	Lang	Description	Symptom
2840	4.0	pgCC	pgCC bug.c causes severe errors	PGCC-ICE error. getsname:
2862	4.0	pgf90	module mymodule integer, public,parameter::n=10 integer, private :: i type, public :: mytype integer :: array(n) end type mytype type, public :: yourtype integer :: array(n) end type yourtype type, public :: histype integer :: array(n) end type histype type(mytype), public, parameter :: myex = mytype ((/(0,i=1,n)/)) type(yourtype), public, parameter :: & yourex = yourtype (myex %array) type(histype), public, parameter ::hisex = histype(0) end module mymodule program main use mymodule write(*,*) myex write(*,*) write(*,*) yourex write(*,*) write(*,*) hisex write(*,*) stop end program main	PGF90-S-0069-Illegal implied DO expression (bug1.f90: 22) PGF90-S-0000-Internal compiler error. _dinit_acl,array 29

2997	5.0	pgcc	#include <search.h>
			char * key1 = "entry1";
			char * key2 = "entry2";
			int main ()
			{
			int size;
			ENTRY entry1;
			ENTRY entry2;
			ENTRY * e;
			size = 100;
			if (!hcreate(size)) {
			printf("error creating hash table\n");
			return 1;
			}
			entry1.key = key1;
			entry2.key = key2;
			e = hsearch(entry1, FIND);
			if (e)
			printf("found %s\n", e->key);
				gcc passes small structs differently on Opteron than larger structs. We have fixed this to be compatible with gcc.
				
				Program seg faults on Opterons.

```
  else
    printf("not found\n");
  e = hsearch(entry1, ENTER);
  e = hsearch(entry2, ENTER);
  e = hsearch(entry1, FIND);
if(e)
  printf("found %s\n", e->key);
  e = hsearch(entry2, FIND);
  if (e)
 printf("found %s\n", e->key);
  hdestroy();return 0;
}
```

3005	5.0	pgf77	users program worked with g77, not with pgf77.	-mcmodel=medium failing
3007	5.0	all	openmp programs properly warn when cpu count is less than thread count. MP_WARN flag prevents the message	Annoying warning message
3009	5.0	pgcc	Problem with programs built by inlining alloca.	GCC test fails with -Minline
3011	5.0	pgf90	program test character*4 C50 character c*4(50) end	The c*4(50) format was not accepted.
3022	5.0	fortran	problems with -mcmodel=medium	seg faults,

3028	5.0	pgcc	#include <stdlib.h>
			> #include <time.h>
			#include <sys/types.h>
			#include <sys/times.h>
			void main() {
			clock_t elp;
			ldiv_t res;
			struct tms itim;
			elp += times(&itim);
			res = ldiv(elp, CLK_TCK);
			}
				seg faults due to small structs

3034	5.0	pgCC	#include <stdio.h>
			struct {
			void Add_Unique(int gn)
			{printf ("gn = %d\n", gn);}
			} node_connectivity[1];
			struct {
			int Global_Number ()
			{return 6067;}
			int Index () {return 0;}
			} inode;
			static int inverse_map[1] =
			{0};
			main () {
			int
			iv=inverse_map[inode.Index()];
			if(iv>=0)
			node_connectivity[iv].Add_Uniqu
			e(inode.Global_Number());
			}
				seg faults when compiled
				pgCC -O2 x.C

3035	5.0	pgf90	integer in,iam
			logical doit
			doit=.true.
			in=1
			iam=1
			IN1=1
			if(in.eq.IN1 .or. iam.ne.0
			.or. .not.doit)then
			Print *,'in loop 1'
			endif
			if(.not.doit .or. in.eq.IN1
				Progra,m fails if compiled with -i8

			.or. iam.ne.0)then
			Print *,'in loop 2'
			endif
			jj=0
			if(in.eq.IN1) jj=1
			if(iam.ne.0) jj=2
			if(.not.doit) jj=3
			if(jj.ne.0)Print*,'in loop 3'
			stop
			end	

3037	5.0	pgf90	Added new f95 syntax to external stmt external xx,yy !f90 syntax external :: xx,yy ! f95 syntax	reported error on second format.

3038	4.1	pgcc	#include <stdio.h>
			void main() {
			int guard;
			unsigned char next,val;
			guard = 0;
			val = 0x80;
			next = val << 1;
			if (next) {guard = 1;}
			printf("%d\n", guard);
			}
				pgcc –O1 foo.c
				prints '1'
				pgcc –O0 foo.c
				prints '0'

3039	5.0	pgf77	program fails with pgf77 -fast	ICE when compiled pgf77 –fast bad.f
3041	5.0	fortran	Function IRTyp(W) Implicit Real*8(A-H,O-Z) Dimension W(3,4), IOrd(5) Save IOrd, One Data IOrd/2,3,4,6,1/, One/1.d0/ Call TrDet(TrI,DetI,W) ITr = ANInt(TrI) Det = ANInt(DetI) IS = Sign(One,Det) IRTyp = IS*IOrd(IS*ITr+2) End	ICE with –fastsse –tp p7
3042	5.0	pgf90	PROGRAM testing implicit none structure /mystruct/ integer*4 i1,i2,i3,i4,toomuch end structure record /mystruct/ s1,t1,t2 common /t_cmn/ t1,t2 s1.i1 = 101 s1.i2 = 102 t1 = s1 t2 = t1 print *, 'out:', s1.i1, t1.i1, t2.i1 END	Output differs when compiled with or without -fpic
3043	4.1	pgf90	program caused ICE	%pgf90 -c -O BHS.f Lowering Error: symbol sinv$sd is a member reference
3048	5.0	pgf90	64-bit pgcc sizeof(sizeof()) should return 8	sizeof(sizeof()) returns 4
3049	5.0-b	pgf90	program foo integer(8) :: i real(8), dimension(1000) :: x x(:) = 1.0 print *, (x(i),i=1,400) end	implicit do loop failed with integer*8 index

3051	5.0	pgf90	CASE stmt with DEFAULT not last case fails.	MCNP5 fails
3053	5.0	pgf90	solved by 3041	ICE (stack87pos)
3059	5.0	pgf90	MODULE pure_sub_mod IMPLICIT NONE CONTAINS PURE SUBROUTINE pure_sub(a) IMPLICIT NONE INTEGER, INTENT(INOUT) :: a a=a*2 END SUBROUTINE pure_sub END MODULE pure_sub_mod MODULE module_mod IMPLICIT NONE CONTAINS PURE SUBROUTINE another_pure_sub(b,pure_sub) IMPLICIT NONE INTEGER, INTENT(INOUT) :: b INTERFACE PURE SUBROUTINE pure_sub(a) IMPLICIT NONE INTEGER, INTENT(INOUT) :: a END SUBROUTINE pure_sub END INTERFACE CALL pure_sub(b) b=b*2 END SUBROUTINE another_pure_sub END MODULE module_mod PROGRAM test_PURE_error_prog USE module mod,ONLY: another_pure_sub USE pure_sub_mod,ONLY: pure_sub IMPLICIT NONE INTEGER :: a a=2 CALL another_pure_sub(a,pure_sub) PRINT *,a END PROGRAM test_PURE_error_prog	ICE - alignment
3060	5.0	pgf90	Correct DWARF information.	bad debug information
3062	5.0	pgf90	Correct the DWARF information	bad debug information
3063	5.0	pgf90	Correct the DWARF information	Bad debug information
3064	5.0	pgf90	Correct the DWARF information	Bad debug information
3066	5.0	pgf77	NPB LU fails with –fast . –Munroll is the problem.	program fails to compile -fast
3067	5.0	64 MPI	NPB in CDK fails with CLASS=B	mpich 64-bit fails.
3068	5.0	pgCC	program fials to compile	compile failure
3069	5.0	pgf90	POP from lanl program fails.to compile tavg.f90	compile failure
3075	4.1	pgf90	module tpg_mod implicit none public integer,parameter,private::X1=5	bad answers from trim()..

			contains
			subroutine outer()
			character(LEN=X1),dimension(:),
			pointer :: plist
			integer :: ierr
			allocate(plist(8),STAT=ierr)
			if (ierr.ne.0) then
			write(6,*)'did not alloc plist'
			stop
			endif
			plist(1) = "ONE "
			plist(2) = "TWO " –and so on
			plist(8) = "EIGHT"
			call inner(plist(3:5))
			end subroutine outer
			subroutine inner(list)
			character(LEN=X1),
			dimension(:),intent(in)::list
			integer :: i
			do i=1,size(list)
			write(6,*) trim(list(i))
			enddo
			end subroutine inner
			end module tpg_mod
			program test
			use tpg_mod
			implicit none
			call outer
			end program test	

3076	5.0	pgf90	module foo_module integer dummy interface subroutine assumed_shape6(xx) integer five, negfour common /bounds/ five, negfour integer*4 xx(:,negfour:) end subroutine assumed_shape6 end interface end module foo_module program tx_f90_arrays use foo_module integer*4,dim(20,10) ::array5 integer i,j,k,l integer five, negfour common /bounds/ five, negfour do i = 0,19 do j = 0,9 array5(i+1,j+1) = j*20 + i + 1 end do end do call assumed_shape6(array5) end program tx_f90_arrays subroutine assumed_shape6(xxx) integer five, negfour common /bounds/ five, negfour integer*4 xxx(:,negfour:) print*,lbound(xx,1),ubound(xx,1) print*,lbound(xx,2),ubound(xx,2) end subroutine assumed_shape6	ICE pgf90 -g
3082	5.0	pgf90	fails with –Mvect-sse	ICE

3.8 5.1-3 Problems/Limitations

What follows are some known problems and possible suggestions that users may encounter. The FAQs at http://www.pgroup.com/faq/index.htm will provide more up to date information.

Subject	Description	Correction/Workaround
Building MM5	Users report MM5 execution problems.	If *libpgthread.so* IS NOT a link to *libpthread.so*, set **MPSTKZ** to 256M or larger if it IS a link, limit stacksize 256M or larger If compiling –fast or –fastsse, add the switch ***–Mnolre*** For compiling with ***linux86-64,*** add the define -DDEC_ALPHA to your config file so that pointers are stored in integer*8 variables rather than integer*4 variables.
flexlm license for pgCC	license features can no longer be case sensitive. The feature lines of the **pgCC** product is now shown as **pgcpp.** As a result, 5.1 licenses will NOT work with pre-5.1 pgCC compilers.	We have created new versions of pgCC for release 4.1 and 5.0, that will accept the 5.1 license. Contact *trs@pgroup.com* for information.
Switch reminder	All language switches try to adhere to the gnu	Be careful. -fast –Mnolre is different

<table>
<tr><td></td><td>convention that they are executed left-to-right in the order they appear.</td><td>from
-Mnolre –fast, for example.</td></tr>
<tr><td>pgf90 -Mipa</td><td colspan="2">-Mipa=shape propagates shape information for assumed-shape arrays;
-Mipa=shape is part of
-Mipa=fast, but can be disabled with
-Mipa=fast,noshape.
It will erroneously propagate shape information
even when the shape is not constant across multiple calls.
Example:
<pre>   subroutine s(x)
      real x(:,:)
    end subroutine
    subroutine y
     real y(10,10),z(20,20)
    call s(y)
     call s(z)
    end subroutine</pre>
-Mipa=const propagates constant array bounds into adjustable array bounds. If the adjustable array is used in an array assignment, the compiler might generate bad code.
Example:
<pre>   subroutine s(x,n)
     real x(n,n)
     x = 0.0
    end subroutine
    subroutine y
     real y(10,10)
     call s(y,10)

    end subroutine</pre></td></tr>
</table>

3.9 AMD64 Large Array Support

Support for the *medium memory model* in the *linux86-64* environment is provided, with some limitations due to the *PGI Workstation 5.1* compilers

and some limitations that are inherent to the medium memory model itself.

The *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 second 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.

The *medium memory model* allows for larger than 2GB data areas, or *.bss* sections. Program code for the *medium memory model* must be compiled *–fPIC*, or position-independent, and will require additional instructions to reference memory. The effect on performance is a function of the data-use of the application.

The *linux86-64* environment provides static lib*xxx*.**a** archive libraries that are built without *–fPIC*, and dynamic lib*xxx*.**so** shared object libraries that are compiled *–fPIC*. The *–mcmodel=medium* linker switch implies the *–fPIC* switch and will utilize the shared libraries by default. Similarly, the *$PGI/linux86-64/5.1/lib* directory contains the libraries for building *small memory model* codes, and the *$PGI/linux86-64/5.1/libso* directory contains shared libraries for building *–mcmodel=medium* and *–fPIC* executables. *Note*: It appears from the GNU tools and documentation that creation of *medium memory model* shared libraries is not supported. However, you can create static archive libraries (.a) that are *–fPIC*.

3.9.1 Practical Limitations of –mcmodel=medium

The 64-bit address capability of the AMD64 technology can cause unexpected problems when data sizes are enlarged significantly. For example:

- Initializing a large array with a data statement may result in a very large assembly file and object file, where a line of assembler source is required for each element in the array initialized. Code compilation and linking will be very time consuming as well. To avoid this time and space-consuming problem, consider initializing large arrays in the program area in a loop rather than in the declaration.

- Stack space can be problem for data that is stack based. *limit stacksize unlimited* can be used to 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 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 users should take reasonable care to determine if a program failure is due a compiler limitation or an operating system limitation.

3.9.2 Compiler Limitations of –mcmodel=medium

There are a number of limitations on large arrays that are not due to the medium memory model, but are due to compiler limitations. Some of these limitations are common between the GNU compilers (*gcc* and *g77*) and the *PGI Workstation 5.1* compilers, and some are specific to the *PGI Workstation 5.1* compilers.

1. Individual (static) data objects are still limited to less than 2GB in size in *PGF90*, *PGHPF* and *PGC++*. *PGF77* and *PGCC* now support static data objects larger than 2GB. F77 applications that use single data objects larger than 2GB must be compiled with *PGF77* using *both* the *–mcmodel=medium* and *–Mlarge_arrays* options. C applications with large data objects need only be compiled with *PGCC* using *–mcmodel=medium*. Support for single data objects larger than 2GB will be added to the *PGF90*,

PGHPF and *PGC++* compilers in a future release.

2. *PGF90* treats all local static data in a program as one collective data object (it is aggregated into a single ELF section). In PGI Workstation 5.1, section sizes in PGF90-compiled programs are still limited to be less than 2GB in size. To utilize multiple very large static arrays (aggregate greater than 2GB) using *PGF90*, you will need to assign the large arrays to distinct COMMONs. This limitation will be removed in a future release of the PGI compilers and tools. As of Release 5.1, this limitation no longer applies to *PGF77*.

3. Dynamically allocated arrays in *C* can be larger than 2GB, using the `malloc()` operation to return a pointer in either *PGCC* or *gcc*. *PGF90*, however, cannot `ALLOCATE` individual arrays of size greater than 2GB. NOTE: no compile-time or pre-specified run-time error will occur if you attempt to `ALLOCATE` an array larger than 2GB using *PGF90*. 32-bit truncation will usually mask the problems you encounter. If you pass a pointer to an array larger than 2GB (for example by returning it to a Fortran program unit from a called *C* program unit), you must access it very carefully. Usually, the array should be referenced as if it is a one-dimensional array. All referencing of multiple dimension arrays within Fortran is restricted to a 32-bit index derived from the individual indices of the array. Indexing into a dynamically allocated 1-dimensional array, where the index is declared `INTEGER*8`, will be successfully evaluated as a 64-bit address.

3.9.3 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 less than 1GB 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 5.1*:

```
% 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 5.1 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
```

3.9.4 Large Array Example in Fortran

The following example needs preprocessing, and illustrates the major difference between pgf90 and pgf77 in the 5.1 release. pgf90 and pgf77 both use 64-bit addresses when compiled *–mcmodel=medium*, but only pgf77 allows for 64-bit integer index support.

Consider the following example:

```
% cat matadd.f
      program matadd
      integer i, j, k, size, l, m, n
#if define (USE_PGF90)
      parameter (size=13000) ! 1GB<size<2GB
#else
     parameter (size=16000) ! >2GB
#endif
      parameter (m=size,n=size)
      real*8 a(m,n),b(m,n),c(m,n),d
#if define (USE_PGF90)
      common/aa/a
      common/bb/b
      common/cc/c
#endif
      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 *PGF90* using *–mcmodel=medium*:

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

The *PGF90* compiler places all local static data for a program in a single section of the generated executable, and therefore the combined sizes exceed the 2GB limit on any single section (in this case *.bss*) generated by *PGF90*. This limitation is handled by

```
      common/aa/a
      common/bb/b
      common/cc/c
```

after *a*, *b*, and *c* are first declared. You can compile and execute as follows:

```
% pgf90 -mp -o matadd -DUSE_PGF90 -Mpreprocess matadd.f
-mcmodel=medium
% setenv OMP_NUM_THREADS 2
% matadd
M =        13000 , N =        13000
c(M,N) =     390026000.0000000
```

For compiling with pgf77, we enlarge the array dimensions to create (>2GB) a very large array, and compile with

```
% pgf77 -mp -o matadd matadd.f -mcmodel=medium
-Mlarge_arrays -fast
% 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 ran about 33% faster with OMP_NUM_THREADS set to 2, instead of 1.

3.10 The PGI CDK 5.1 and *libpthread*

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, we are seeing libpthread.so and libpthread.a with 're-sizeable' thread stack areas. In these cases

1. The filename $PGI/linux86/5.1/lib/libpgthread.so is a soft link to /usr/lib/libpthread.so.

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

3.11 The PGI CDK 5.1 and *glibc*

Release 5.1 of the *PGI CDK* compilers and tools for *linux86* are built and validated under both the Linux 2.2.10 through 2.4.x kernels. Distributions of Linux, from Red Hat 7.0 to 9.0 and SuSE 7.1 to 9.0, incorporate revision 2.2.10 or greater of the Linux kernel and *glibc2.2.x* or greater. If you are using a version of Linux that is supported by the 5.1 CDK release, the PGI installation script will automatically detect it. Your installation will be modified as appropriate for these systems. While attempts are made to handle mixtures of *glibc* versions and Linux releases, we may not correctly install on a customized configuration, or work successfully because of it. We do test the typical Linux configurations with glibc as they come from the box.

3.12 The PGI ACML, BLAS & LAPACK *Libs*

Precompiled versions of the ACML, BLAS and LAPACK math libraries are included. libacml.a and libacml.so are included for AMD64 cpus and Pentium 4 cpus with SSE/SSE2 instructions. libblas and liblapack are available for all cpu types, and –FPIC versions for AMD64 machines.

A source for additional LAPACK documentation can be found at http://www.cs.colorado.edu/~lapack .

3.13 OpenMP Tutorial

A self-guided online tutorial is available to help you become familiar with how OpenMP parallelization directives. In particular, the tutorial takes the user step by step through the process of parallelizing the NAS FT benchmark using OpenMP directives. The tutorial can be found at:

```
ftp://ftp.pgroup.com/pub/SMP
```

You can download this file using a web browser, and unpack the file using the following commands:

```
% gunzip fftpde.tar.gz
% tar xvf fftpde.tar
```

Change directories to the `fftpde` sub-directory, and follow the instructions in the README file.

3.14 Debugging with PGDBG

4. **pgdbg 5.1, like the compilers, comes in two versions: *linux86* and *linux86-64* . On the** AMD64 **Technology systems running a linux86-64 environment, users can debug a 32-bit application and a 64-bitapplication on the same system, and the same process and thread controls are available. The notes below apply to both versions. Only the *linux86-64* environment can run the *linux86-64* version of *pgdbg*.** address.

3.14.1 PGDBG 5.1 Features

Note: Most of this information was also in the PGI CDK 4.0 Release Notes. It is present here and will be part of the PGDBG User's Guide in a future release, and will not be removed from the release notes until it has. PGDBG has had a number of corrections, and it now is supported under ssh, but beyond that the features have not changed from 4.0.

PGDBG 5.1 can debug SMP OpenMP (or Linux pthread) programs, as well as multiprocess cluster programs executed via *mpirun*. The PGI license file restricts the total number of threads and processes that PGDBG will debug.

PGDBG 5.1 supports ***ssh*** as well as ***rsh*** . A new environment variable, **PGRSH**, should be set to ***ssh*** or ***rsh***, to indicate the communication needed.

PGDBG's parallel debug capabilities are extensively documented at http://www.pgroup.com/docs.htm or at $PGI/doc/index.htm. This documentation is intended to supplement Chapter 15 of the PGI User's Guide.

The following enhancements are included in PGDBG 5.1:

- Combined Multi-process and Multi-thread Support
- SSH support
  - be sure to 'export PGRSH=ssh'
- Multi-process Support
  - Process Ids obtained from *mpirun*
  - Same source and debug info used for all processes
  - full process control
  - process grouping
  - informative messages regarding state and location
- Process Control
  - concise control of groups of processes

- process synchronization
- configurable process stop and wait modes
- serial, and process-only debug modes

- OpenMP & Linuxthread Support
  - threads identified by OpenMP logical CPU ID
  - automatic thread detection and attach
  - full thread control in parallel regions
  - thread grouping
  - line level debugging preserved when a thread
    - enters a parallel region
    - enters a serial region
    - hits an OpenMP barrier
    - hits an OpenMP synchronize statement
    - enters an OpenMP sections program section
  - informative messages regarding thread state and location

- Thread Control
  - concise control of groups of threads
  - thread synchronization
  - configurable thread stop and wait modes
  - serial, and threads-only debug modes

- GUI Enhancements
  - Thread sub-window. Lists each thread by its logical CPU ID. Displays for each thread its state and stop location. Threads are grouped by parent process.
  - Program I/O sub-window. Pops up automatically when program prints to stdout. The program I/O sub-window can also be raised from the Window menu.
  - Output written to stdout by the process being debugged is no longer block buffered.
  - process grid. Displays each process as a color coded button in a grid. Click on a grid element to refresh the GUI in the scope of that process. Each grid element is numbered with the process's logical ID.
  - Process grouping. Control processes in groups
  - Thread grid. Displays each thread as a color coded

button in a grid. Click on a grid element to refresh the GUI in the scope of that thread. Each grid element is numbered with the thread's logical CPU ID.
    - Thread grouping. Control threads in groups.
- Other Enhancements

5. Better support for Fortran arrays and pointersaddress.

3.14.2 PGDBG 5.1 Technical Information

Here are a number of details not documented in the PGDBG User's Guide.

3.14.2.1 Threads and Signals

PGDBG intercepts all signals sent to any of the threads in a multi-threaded program, and passes them on according to that signal's disposition maintained by *PGDBG* (see the ***catch***, ***ignore*** commands).

If a thread runs into a busy loop, or if the program runs into deadlock, control-C over the debugging command line to interrupt the threads. This causes SIGINT to be sent to all threads. By default *PGDBG* does not relay SIGINT to any of the threads, so in most cases program behavior is not affected.

Sending a SIGINT (control-C) to a program while it is in the middle of initializing its threads (calling ***omp_set_num_threads()***, or entering a parallel region) may kill some of the threads if the signal is sent before each thread is fully initialized. Avoid sending SIGINT in these situations. When the number of threads employed by a program is large, thread initialization may take a while.

3.14.2.2 Signals Used by Internally by PGDBG

SIGTRAP indicates a breakpoint has been hit. A message is displayed whenever a thread hits a breakpoint. SIGSTOP is used internally by

PGDBG. Its use is mostly invisible to the user. Changing the disposition of these signals in *PGDBG* will result in undefined behavior.

6. Reserved Signals: On linux86, the thread library uses SIGRT1, SIGRT3 to communicate among threads internally. In the absence of real-time signals in the kernel, SIGUSR1, SIGUSR2 are used. Changing the disposition of these signals in *PGDBG* will result in undefined behavior.

3.14.3 Scoping

Nested Subroutines

To reference a nested subroutine you must qualify its name with the name of its enclosing function using the scoping operator @.

For example:

```
subroutine subtest (ndim)
integer(4), intent(in) :: ndim
integer, dimension(ndim) :: ijk
call subsubtest ()
contains
    subroutine  subsubtest ()
    integer :: I
    i=9
    ijk(1) = 1
    end subroutine subsubtest
    subroutine  subsubtest2 ()
    ijk(1) = 1
    end subroutine subsubtest2
end subroutine subtest
program testscope
integer(4), parameter :: ndim = 4
call subtest (ndim)
end program testscope
```

```
pgdbg> break subtest@subsubtest
breakpoint set at: subsubtest line: 8 in "ex.f90" address:
0x80494091
pgdbg> names subtest@subsubtest
i = 0
pgdbg> decls subtest@subsubtest
arguments:
variables:
integer*4 i;
pgdbg> whereis subsubtest
function:       "ex.f90"@subtest@subsubtest
```

Fortran 90 Modules

To access a member ***mm*** of a Fortran 90 module ***M*** you must qualify ***mm*** with ***M*** using the scoping operator **@**. If the current scope is ***M*** the qualification can be omitted.

For example:

```
module M
implicit none
real mm
contains
subroutine stub
print *,mm
end subroutine stub
end module M
```

```
program test
use M
implicit none
call stub()
print *,mm
end program test
```

7. pgdbg> Stopped at 0x80494e3, function MAIN, file M.f90, line 13
   #13: call stub()
   pgdbg> which mm
   "M.f90"@m@mm
   pgdbg> print "M.f90"@m@mm
   0
   pgdbg> names m
   mm = 0
   stub = "M.f90"@m@stub
   pgdbg> decls m
   real*4 mm;
   subroutine stub();
   pgdbg> print m@mm
   0
   pgdbg> break stub
   breakpoint set at: stub line:6 in "M.f90" address: 0x8049446 1
   pgdbg> c
   Stopped at 0x8049446, function stub, file M.f90, line 6
   Warning: Source file M.f90 has been modified more recently than object file
   #6: print *,mm
   pgdbg> print mm
   0
   pgdbg>

3.14.4 Lexical Blocks

Line numbers are used to name lexical blocks. The line number of the first instruction contained by a lexical block indicates the start scope of the lexical block.

Below variable ***var*** is declared in the lexical block starting at line 5. The lexical block has the unique name ***"lex.c"@main@5***. The variable ***var*** declared in ***"lex.c"@main@5*** has the unique name ***"lex.c"@main@5@var***.

For Example:

```
lex.c:
main()
{
    int var = 0;
    {
        int var = 1;
        printf("var %d\n",var);
    }
    printf("var %d\n",var)
}
```

8. pgdbg> n
   Stopped at 0x8048b10, function main, file /home/pete/pgdbg/bugs/workon3/ctest/lex.c, line 6
   #6: printf("var %d\n",var);
   pgdbg> print var
   1
   pgdbg> which var
   "lex.c"@main@5@var
   pgdbg> whereis var
   variable: "lex.c"@main@var
   variable: "lex.c"@main@5@var
   pgdbg> names "lex.c"@main@5
   var = 1

3.14.5 Private Variables

PGDBG understands private variables with some restrictions. In particular, inspecting private variables while debugging FORTRAN programs is not supported.

Private variables in *C* must be declared in the enclosing lexical block of the parallel region in order for them to be visible using *PGDBG*.

For example:

```
{
    #pragma omp parallel
    {
        int i;
        ...
        /* i is private to 'this' thread */
        ...
    }
}
```

In the above case, `i` would be visible inside *PGDBG* for each thread. However, in the following example, `i` is not visible inside *PGDBG*:

```
{
    int i;
    #pragma omp parallel private(i)
    {
        ...
        /* i is private to 'this' thread
           but not visible within PGDBG */
        ...
    }
}
```

9. A private variable of a Thread A is accessed by switching the current thread to A, and by using the name (qualified if necessary) of the private variable.

3.14.6 Graphical User Interface (GUI) Notes

3.14.6.1 Setting the Font

Use the ***xlsfonts*** command to list all fonts installed on your system, then choose one you like. For this example, we choose a *sony* font that is completely specified by the following string:

```
-sony-fixed-medium-r-normal—24-230-75-75-c-120-iso8859-
1
```

There are two ways to set the font that your *PGDBG* GUI uses.

1. Use your .Xresources file:

```
Xpgdbg*font : <chosen font>
pgdbg*font : <chosen font>
```

For example:

```
pgdbg*font : -sony-fixed-medium-r-normal--24-230-75-75-
c-120-iso8859-1
```

You will have to merge these changes into your X environment for them to take effect. You can use the following command:

```
% xrdb -merge $HOME/.Xresources
```

2. Use the command line options : ***–fn <font>***. For example:

% pgdbg -fn -sony-fixed-medium-r-normal--0-0-100-100-c-0-jisx0201.1976-0...

3.14.6.2 Control-C from GUI

The active window must be the command window (upper window) where the *PGDBG* prompt appears for control-C to interrupt the program being debugged. interrupt the program being debugged.

3.14.6.3 Shared Object Files

PGDBG supports debugging of dynamically linked executables that reference shared object files created using the compilers. If the executable being debugged is dynamically linked, *PGDBG* will report when each shared object is loaded and/or unloaded.

For example:

```
pgdbg> ...
pgdbg> n
Stopped at 0x8048bee, function main, file
dynload.c, line 36
#36: handle = dlopen("libpetesSO2.so",RTLD_NOW);
pgdbg> n
libpetesSO2.so loaded by ld-linux.so.2.
Stopped at 0x8048c31, function main, file
dynload.c, line 41
#41:       if (handle){
pgdbg> n
Stopped at 0x8048c37, function main, file
dynload.c, line 42
#42:         dlclose(handle);
pgdbg> n
libpetesSO2.so unloaded by ld-linux.so.2.
Stopped at 0x8048c42, function main, file
dynload.c, line 45
#45:     }
pgdbg> ...
```

The global symbols defined by a dynamically linked shared object are visible during a *PGDBG* debug session. These symbols are currently available only without type and line number information. The machine level *PGDBG* commands (`breaki`, `dump`, `hwatch`, `disasm`, etc) are useful for inspecting these symbols. Each symbol is available with respect to the load status of its defining shared object.

For example, dynamically-linkable Position Independent Code (PIC) is implemented using a Procedure Linkage Table (PLT) and Global Offset Table (GOT). Each PIC function is bound lazily at run-time. If a function has not been linked dynamically, *PGDBG* reports the address of its PLT

entry as its address. If a function has been linked dynamically, *PGDBG* reports the virtual address of the function itself. So, *PGDBG* reports the current or "effective" address of symbols with respect to dynamic linking and loading. *PGDBG* treats global symbols defined in shared objects in a similar way. The address of a global variable may be the address of its GOT entry or an absolute address, depending in part on its load status.

4 PGHPF 5.1

PGHPF 5.1 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 for a list of features, which is a continuation of the 5.1 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*.

4.1 Summary of Changes

The following features have been added to *PGHPF 5.1* 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 5.1 – see section 3.12.1 for more information on this feature

4.2 Restrictions

This section lists Fortran 90 and HPF features that are *not* supported in *PGHPF 5.1*.

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 5.1* 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 5.1* 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.

5 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
```