PGI® Server 6.0
PGI® Workstation 6.0

Installation &
Release Notes

The Portland Group™
STMicroelectronics, Inc
Two Centerpointe Drive
Lake Oswego, OR 97035
www.pgroup.com

PGI Server 6.0 / PGI Workstation 6.0
Installation & Release Notes

First Printing: Release 6.0-2, March, 2005
Second Printing: Release 6.0-5, July, 2005
Third Printing: Release 6.0-8, November, 2005

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

Table of Contents

1 PGI Release 6.0 Introduction

Welcome to Release 6.0 of *PGI Workstation* and *PGI Server*, a set of Fortran, C and C++ compilers and development tools for 32-bit and 64-bit *x86*-compatible processor-based workstations and servers running versions of the Linux* (32-bit and 64-bit) and Windows* (32-bit only) operating systems.

All workstation-class compilers and tools products from The Portland Group (*PGHPF Workstation*, for example) are subsets of the *PGI Workstation* product. These workstation-class products provide for a node-locked single-user license, meaning one user at a time can compile on the system on which the *PGI Workstation* compilers and tools are installed.

PGI Server products are offered in configurations identical to the workstation-class products, but provide for network-floating multi-user licenses. This means that two or more users can use the *PGI* compilers and tools concurrently on any compatible system networked to the system on which the *PGI Server* compilers are installed.

These release notes apply to all Workstation-class and Server-class compiler products from The Portland Group.

1.1 Product Overview

Release 6.0 of *PGI Workstation* and *PGI Server* includes the following

components:

- *PGF95* OpenMP* and auto-parallelizing Fortran 90/95 compiler
- *PGF77* OpenMP and auto-parallelizing FORTRAN 77 compiler
- *PGHPF* data parallel High Performance Fortran compiler
- *PGCC* OpenMP and auto-parallelizing ANSI and K&R *C* compiler
- *PGC++* OpenMP and auto-parallelizing ANSI *C++* compiler.
  NOTE: *PGC++ is not supported on Win32 platforms.*
- *PGPROF* graphical OpenMP/multi-thread performance profiler
- *PGDBG* graphical OpenMP/multi-thread symbolic debugger.
  NOTE: *PGDBG is not supported on Win32 platforms.*
- Online documentation in PDF, HTML and `man` page formats.
- A UNIX*-like shell environment for *Win32* platforms.

Depending on the product configuration you purchased, you may not have licensed all of the above components.

1.2 Terms and Definitions

Following are definitions of terms used in the context of these release notes.

driver – the compiler *driver* controls the compiler, linker, and assembler and adds objects and libraries to create an executable. The *–dryrun* option illustrates operation of the driver. `pgf77`, `pgf95`, `pghpf`, `pgcc`, and `pgCC` are drivers for the PGI compilers. A `pgf90` driver is retained for compatibility with existing makefiles, even though `pgf90` and `pgf95` are identical.

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

x87 – 80-bit IEEE stack-based floating-point unit (FPU) and associated

instructions on *x86*–compatible CPUs.

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

AMD64 – a 64-bit processor from AMD designed to be binary compatible with 32-bit *x86* processors, and incorporating new features such as additional registers and 64-bit addressing support for improved performance and greatly increased memory range. This includes the AMD* Athlon64* and AMD Opteron* processors.

EM64T – a 64-bit IA32 processor with *Extended Memory 64-bit Technology* extensions designed to be binary compatible with AMD64 processors. This includes newer generation Intel Pentium 4 and Intel Xeon processors. Most comments in these release notes that apply to AMD64 technology processors also apply to IA32 processors with EM64T extensions.

SSE1 - 32-bit IEEE 754 FPU and associated *streaming SIMD extensions* (SSE) instructions on Pentium III, AthlonXP* and later 32-bit *x86*, *AMD64* and *EM64T* compatible CPUs, enabling scalar and packed vector arithmetic on single-precision floating-point data

SSE2 – 64-bit IEEE 754 FPU and associated SSE instructions on P4/Xeon and later 32-bit *x86*, *AMD64* and *EM64T* compatible CPUs, enabling scalar and packed vector arithmetic on double-precision floating-point data

SSE3 – additional 32-bit and 64-bit SSE instructions to enable more efficient support of arithmetic on complex floating-point data on 32-bit *x86*, *AMD64* and *EM64T* compatible CPUs with so-called *Prescott New Instructions* (PNI), such as Intel IA32 processors with EM64T extensions and newer generation (Revision E and beyond) AMD64 processors.

linux86 – 32-bit Linux operating system running on an *x86, AMD64* or *EM64T* processor-based system, with 32-bit GNU tools, utilities and

libraries used by the PGI compilers to assemble and link for 32-bit execution.

Win32 – any of the 32-bit Microsoft* Windows* Operating Systems (XP/2000/Server 2003) running on an *x86, AMD64* or *EM64T* processor-based system. On these targets, the PGI compiler products include additional tools and libraries needed to build executables for 32-bit Windows systems.

linux86-64 – 64-bit Linux operating system running on an *AMD64* or *EM64T* processor-based system, with 64-bit and 32-bit GNU tools, utilities and libraries used by the PGI compilers to assemble and link for execution in either *linux86* or *linux86-64* environments. The 32-bit development tools and execution environment under *linux86-64* are considered a cross development environment for *x86* processor-based applications.

–mcmodel=small – Compiler/linker switch to produce *small memory model* format objects/executables in which both code (*.text*) and data *(.bss*) sections are limited to less than 2GB. This is the default and only possible format for *linux86* 32-bit executables. This is the default format for *linux86-64* executables. Maximum address offset range is 32-bits, and total memory used for OS+Code+Data must be less than 2GB.

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

Large Arrays – Arrays with aggregate size larger than 2GB**,** which requires 64-bit index arithmetic for accesses to elements of arrays. Program units that use *Large Arrays* must be compiled using *–mcmodel=medium*. If *–mcmodel=medium* is not specified, but *–Mlarge_arrays* is specified, the

default small memory model is used but all index arithmetic is performed in 64-bits. This can be a useful mode of execution for certain existing 64-bit applications that use the small memory model but allocate and manage a single contiguous data space larger than 2GB.

Shared library – A library of the form *libxxx.so* containing objects that are dynamically linked into a program at the time of execution.

Static linking – Using *–Bstatic* to ensure all objects are included in the generated executable at link time. Static linking causes objects from static library archives of the form *libxxx.a* to be linked in to your executable, rather than dynamically linking the corresponding *libxxx.so* shared library. Static linking of executables linked using the *–mcmodel=medium* option is now supported.

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

Dual-core – Some AMD64 and EM64T CPUs incorporate two complete processor cores (functional units, registers, level 1 cache, level 2 cache, etc) on a single silicon die. These are referred to as *Dual-core* processors. For purposes of OpenMP, threads, or MPI parallelism, these cores function as 2 distinct processors. However, the two processing cores are on a single chip occupying a single socket on the system motherboard. For purposes of PGI software licensing, one Dual-core processor is treated as a single CPU. In particular, a *PGI Workstation* license that typically limits OpenMP process creations to a maximum of 4, will on a Dual-core system with 4 CPUs enable process creations up to a maximum of 8 (4 CPUs * 2 Cores per CPU or 8 total processes).

NUMA – Non-Uniform Memory Access. A type of multi-processor system architecture in which the memory latency from a given processor to

a given portion of memory can vary, resulting in the possibility for compiler or programming optimizations to ensure frequently accessed data is “close” to a given processor as determined by memory latency.

2 PGI Release 6.0 Installation Notes

2.1 Introduction

Section 2.2 below describes how to install *PGI Workstation* or *PGI Server* in a generic manner on Linux. Section 2.6 describes how to install *PGI Workstation* on Win32 systems. Installations using these instructions do not need to run a license daemon, except as noted below.

The PGI compilers and tools are license-managed. As noted in the sections that follow, generation of permanent license keys is performed using your personalized account on the *http://www.pgroup.com* web page. When you purchase a permanent license, the e-mail order acknowledgement you receive includes complete instructions for logging on to the *pgroup.com* web page and generating permanent license keys.

For products using PGI-style licensing (the default), a single user can run as many simultaneous copies of the compiler as desired, on a single system, and no license daemon or *Ethernet* card is required. However, usage of the *PGI Workstation* compilers and tools is restricted to a pre-specified *username*. If you would like the compilers and tools to be usable under any *username*, or if you are installing a multi-user network-floating *PGI Server* product, you must request FLEXlm*-style license keys when generating your keys and use FLEXlm-style licensing as outlined below.

Installation of FLEXlm-style licensing is more complicated than PGI-style

licensing. If you require FLEXlm-style licensing, you must install the PGI compilers and tools according to the instructions in section 2.2, then install and configure the FLEXlm license management software according to the instructions in section 2.3. Section 2.3 describes how to configure license daemons for Linux, including installation of the license daemon and proper initialization of the `LM_LICENSE_FILE` environment variable. FLEXlm-style licensing is not currently available with PGI compiler products for Win32.

Regardless of the licensing mechanism you choose, when the PGI compilers and tools are first installed they are usable for 15 days without a permanent license key.

NOTE

At the conclusion of the trial period, the PGI compilers and tools and any executable files generated prior to the installation of permanent license keys will cease to function. Any executables, object files, or libraries created using the PGI compilers in demo mode must be recompiled with permanent license keys in place.

Executable files generated with permanent license keys in place are unconstrained, and will run on any compatible system regardless of whether the PGI compilers are installed. However, if you change the configuration of your system by adding or removing hardware, your license key may become invalid. Please contact The Portland Group if you expect to reconfigure your system to ensure that you do not temporarily lose the use of the PGI compilers and tools.

For the first 60 days after your purchase, you may send technical questions about these products to the e-mail address *trs@pgroup.com*. If you have purchased a PGI Software Subscription, you will have access to e-mail support for an additional 12 months and will be notified by e-mail when maintenance releases occur and are available for electronic download and installation. Phone support is not currently available. Contact us at *sales@pgroup.com* if you would like information regarding the

subscription service for the PGI products you have purchased.

2.2 Installing on Linux86 or Linux86-64

If you specify `/usr/pgi` as the base directory for installation, the following directory structure will be created by the PGI installation script:

Name of directory	Contents
`/usr/pgi/linux86/6.0/bin`	*linux86* 32-bit compilers & tools
`/usr/pgi/linux86/6.0/lib`	*linux86* 32-bit runtime libraries
`/usr/pgi/linux86/6.0/liblf`	*linux86* 32-bit large-file support libs (*–Mlfs*)
`/usr/pgi/linux86/6.0/include`	*linux86* 32-bit header files
`/usr/pgi/linux86-64/6.0/bin`	*linux86-64* compilers & tools
`/usr/pgi/linux86-64/6.0/lib`	*linux86-64 –mcmodel=small* libs
`/usr/pgi/linux86-64/6.0/libso`	*linux86-64 –fpic* shared libraries for *–mcmodel=medium* development
`/usr/pgi/linux86-64/6.0/include`	*linux86-64* header files
`/usr/pgi/linux86/6.0/REDIST` `/usr/pgi/linux86-64/6.0/REDIST`	Re-distributable runtime libraries
`/usr/pgi/linux86/6.0/EXAMPLES` `/usr/pgi/linux86-64/6.0/EXAMPLES`	Compiler examples
`/usr/pgi/linux86/6.0/doc` `/usr/pgi/linux86-64/6.0/doc`	Documentation
`/usr/pgi/linux86/6.0/man` `/usr/pgi/linux86-64/6.0/man`	UNIX-style man pages
`/usr/pgi/linux86/6.0/jre` `/usr/pgi/linux86-64/6.0/jre`	JAVA environment for *PGDBG* and *PGPROF* graphical user interfaces
`/usr/pgi/linux86/6.0/src` `/usr/pgi/linux86-64/6.0/src`	*PGHPF* MPI interface file, *mpi.c*

For installations on 32-bit *x86* systems, the PGI installation script installs only the *linux86* versions of the PGI compilers and tools. For installations on 64-bit *x86* systems running a *linux86-64* execution and development environment, the PGI installation script will attempt to install both the *linux86* and *linux86-64* versions of the PGI compilers and tools. The 32-bit and 64-bit compilers, tools and supporting components have the same command 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 settings.

Bring up a shell command window on your system. The instructions below assume you are using *csh*, *sh*, *ksh*, *bash*, 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 four. The PGI products should fit into less than 250 MB of disk space. For *linux86-64* installations, assuming double the disk space requirements (500 MB) is sufficient.

Step 1 – If you received this software on a CD-ROM, please skip to step 2. If you downloaded the software from `http://www.pgroup.com` or another electronic distribution site, then in the instructions that follow, `<tarfile>` needs to be replaced with the name of the file that was downloaded.

The PGI products cannot be installed into the same directory where the tar file is unpacked. Unpack the tar file in a temporary directory before installation:

```
% mkdir /tmp/pgi
% mv <tarfile>.tar.gz /tmp/pgi
% cd /tmp/pgi
% tar xpfz <tarfile>.tar.gz
```

Step 2 – The install script ***must*** be run to properly install the software. If you downloaded the software from the Internet, execute the following script in the directory where you unpacked the tar file:

```
% ./install
```

If you are installing from a CD-ROM, issue the following command:

```
% /mnt/cdrom/install
```

The install script will list the products that are available on the CD-ROM or in the download package. You will be asked which products should be installed and to select an installation directory. After the software is installed, the script will do some system-specific customization and then initialize the licensing, which is covered in step 3 below.

NOTE: If you have difficulty running this script, especially on a Slackware Linux system, check the permissions on `/dev/null`. Permission should be set to "`crw-rw-rw-`". Reset permissions to this value if necessary – super-user permissions are required.

NOTE: some systems use a CD-ROM volume manager that may insert an additional directory in the above pathname. For example, the pathname might be

```
% /cdrom/pgisoft/install
```

If you are not sure how to access the CD-ROM drive, check with your system administrator.

Step 3 – All of the PGI compilers and tools are license-managed. *PGI Workstation* products that are node-locked and limited to a single user have no need to run a license daemon. If you want the *PGI Workstation* compilers to be usable by any one user rather than locked to a specific username, or if you are installing a multi-user *PGI Server* product, you must use FLEXlm and must specifically request FLEXlm-style keys when generating license keys over the PGI web page at *http://www.pgroup.com/support/keylogin.htm*. If you have purchased the compilers and tools that you are installing, you should have received an order acknowledgement e-mail with instructions on how to generate your license keys through the *pgroup.com* web page. *Note*: FLEXlm-style licensing of the *PGI Workstation* products is not available on Win32 systems.

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: 6.0-5
```

The message above is also saved to the file `/usr/pgi/license.info` for retrieval at a later time.

Once you have obtained your permanent license keys using your personalized account on the *pgroup.com* web page, place them in the file `/usr/pgi/license.dat` (substitute the appropriate installation directory path if you have not installed in the default `/usr/pgi` directory). If you want the *PGI Workstation* compilers to be usable by any one user, rather than locked to a specific username, you must use FLEXlm and must specifically request FLEXlm-style license keys using your account on the *pgroup.com* web page.

Step 4 – You can view the online HTML and PDF documentation using any web browser. Assuming you use *Netscape**, issue the following command:

```
% netscape /usr/pgi/linux86/6.0/doc/index.htm
```

You may want to bookmark this location for easy future reference to the online manuals.

Step 5 – With either the temporary or permanent license file in place,

execute the following commands to make the products you have purchased accessible. Note that the path settings below assume that a 32-bit Linux product has been installed.

Assuming `csh` and installation in the default `/usr/pgi` directory:

```
% set path = (/usr/pgi/linux86/6.0/bin $path)
% setenv MANPATH "$MANPATH":/usr/pgi/linux86/6.0/man
```

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

```
% PATH=/usr/pgi/linux86/6.0/bin:$PATH
% export PATH
% MANPATH=$MANPATH:/usr/pgi/linux86/6.0/man
% export MANPATH
```

If you install the *linux86-64* versions of the compilers and wish to target *linux86-64* as the default, use the same setup with an alternate path setting:

```
% set path = (/usr/pgi/linux86-64/6.0/bin $path)
% setenv MANPATH "$MANPATH":/usr/pgi/linux86-64/6.0/man
```

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

```
% PATH=/usr/pgi/linux86-64/6.0/bin:$PATH
% export PATH
% MANPATH=$MANPATH:/usr/pgi/linux86-64/6.0/man
% export MANPATH
```

You should add these commands to your startup files to ensure you have access to the PGI compilers and tools upon future logins.

Step 6 – You can verify the release number of the products you have installed using the *–V* option on any of the compiler commands. If you use *–v* instead, you will also see 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 95, use "`pgf95 -V x.f`"
- For HPF, use "`pghpf -V x.f`"
- For C++, use "`pgCC -V x.c`"
- For ANSI 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.

2.3 Using FLEXlm on Linux

If you want the *PGI Workstation* compilers to be usable by any one user, rather than locked to a specific *username*, or if you are installing a multi-user *PGI Server* product, you must use the FLEXlm software license management system from Macrovision* Software as outlined below.

IMPORTANT NOTE: Release 6.0 includes a newer version of the Macrovision FLEXlm software. The *lmgrd* and *pgroupd* daemons are updated and must be used in preference to versions shipped with previous releases of the PGI products. You can co-install Release 6.0 with Release 5.2, and use both versions of the compilers and tools with a single Release 6.0 license file and the new versions of *lmgrd* and *pgroupd*. You must modify your `/etc/rc.d` file to use the new *lmgrd*, if you use this file to start *lmgrd* automatically after a reboot of your system. For example:

```
## Path to master daemon lmgrd
# Commented out previous path to 5.2:
#LMGRD=$PGI/<target>/5.2/bin/lmgrd
LMGRD=$PGI/<target>/6.0/bin/lmgrd

## Command to stop lmgrd
#Commented out previous path to 5.2:
#LMUTIL=$PGI/<target>/5.2/bin/lmutil
LMUTIL=$PGI/<target>/6.0/bin/lmutil
```

where <target> is replaced appropriately with linux86 or linux86-64. See ***Step 4*** below for complete details on setup and usage of these files.

Step 1 – Install the software as described in section 2.2 above.

Step 2 – Once you have obtained permanent FLEXlm-style license keys (see section 2.2 above, *Step 3*, for how to obtain these), place them in a file named license.dat in the /usr/pgi directory. For example, if you have purchased *PGF77 Workstation* for Linux, the license.dat file should look similar to the following:

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

FEATURE pgf77-linux86 pgroupd 6.000 31-dec-0 1 \
2B9CF0F163159E4ABE32 VENDOR_STRING=107209:16 \
HOSTID=<hostid> ck=49

FEATURE pgprof pgroupd 6.000 31-dec-0 1 \
6BDCE0B12EC19D0909F0 VENDOR_STRING=107209:16 \
HOSTID=<hostid> ck=60
```

<hostname> and <hostid> should match those you submitted, 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 /usr/pgi as defined above.

NOTE: In the feature line component VENDOR_STRING=107209:16, 107209 is the Product ID Number (PIN) for this installation. You will have a similar unique PIN number for your installation. Please include your PIN number when sending mail to us regarding technical support for the products you have purchased.

Step 3 – When the license file is in place, execute the following commands to make the products you have purchased accessible. Issue the following commands to initialize your environment for use of FLEXlm (assuming csh):

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

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

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

You should add these commands to your startup files to ensure you have access to the PGI products upon future logins.

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

Step 4 – You must now start the license manager daemon. Edit the shell script template `/usr/pgi/linux86/6.0/bin/lmgrd.rc`. If you have installed the compiler(s) in a directory other than `/usr/pgi`, substitute the correct installation directory into '/usr/pgi' part on line 3 of the script. Now exit the editor and issue the following command to start the license server and *pgroupd* 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 *pgroupd* 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 /usr/pgi/linux86/6.0/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 /usr/pgi/linux86/6.0/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.

Installation of your FLEXlm-style licensing of our products for Linux is now complete. If you have difficulties with the installation, send e-mail to *trs@pgroup.com* for assistance.

2.4 Non-Linux86 License Servers

If you are using a non-*linux86* or non-*linux86-64* system as the license server for the PGI compilers and tools, only Sun Solaris is supported. Versions of FLEXlm available for license management of the PGI compilers and tools on alternative hosts can be found at *http://www.pgroup.com/support/download_licensing.php*.

2.5 Setting Up Your Environment

Now that you have installed the compilers in, for example, `/usr/pgi`, you must initialize your environment to use the compilers successfully. Assume the license file is in `/usr/pgi/license.dat`, and that the *lmgrd* license manager is running. Each user must issue the following sequence of commands to initialize the shell environment before using the PGI compilers and tools.

In csh,

```
% setenv PGI /usr/pgi
% setenv MANPATH "$MANPATH":$PGI/linux86/6.0/man
% setenv LM_LICENSE_FILE $PGI/license.dat
% set path = ($PGI/linux86/6.0/bin $path)
```

If you have installed both 32-bit and 64-bit PGI compilers and tools and want the 64-bit compilers to be default, replace the last command above with:

```
% set path = ($PGI/linux86-64/6.0/bin $path)
```

Or, assuming bash, sh or ksh:

```
% PGI=/usr/pgi; export PGI
% MANPATH=$MANPATH:$PGI/linux86/6.0/man
% export MANPATH
% LM_LICENSE_FILE=$PGI/license.dat
% export LM_LICENSE_FILE
% PATH=$PGI/linux86/6.0/bin:$PATH
% export PATH
```

or, for 64-bit installations replace the last two commands with:

```
% PATH=$PGI/linux86-64/6.0/bin:$PATH
% export PATH
```

2.6 Installing PGI Workstation 6.0 on Win32

If you are installing *PGI Workstation* from a CD-ROM, insert the CD-ROM into the CD-ROM drive on the system on which the install is to take place. An installation script will automatically be invoked and the installation process will begin. Follow the directions printed to your screen.

If you are installing *PGI Workstation* from the self-extracting file downloaded electronically via ftp, double-click on the `pgiws.exe` file with the left mouse button. The installation process will begin. Follow the

instructions printed to your screen.

As with Linux, the *PGI Workstation* compilers and tools on Win32 are license-managed. However, FLEXlm-style licensing is not available on Win32. All licenses are node-locked. The Win32 serial number is used as the *hostid*. This number will be printed to your screen during the installation process, or can be located by left-clicking on *Start->Settings->Control Panel* (on *Windows XP*, *Start->Control Panel*, and set *classic settings* to get System info) and then double-left-clicking on the *System* icon and left-clicking on the "General" tab. The Win32 serial number will be in the middle of the System Properties window and look something like the following:

```
Registered to:
   <your name>
   <your organization>
   22296-oem-0014072-07487
```

The last number above is the Win32 serial number. Obtain your permanent license keys using your personalized account on the *pgroup.com* web page as outlined in your order acknowledgement, and place them in the file `C:/Program Files/PGI/license.dat` (or specify the appropriate directory path if you have installed in a directory other than the default `C:/Program Files/PGI`). You should now be able to use the PGI compilers and tools from any *PGI Workstation* command window.

2.7 Installation Limitations for Win32

The *PGI Workstation 6.0* release installs in a way that will not corrupt a previous *PGI Workstation* release. However, a new installation of *PGI Workstation 5.2* for Win32 is required to use both installations concurrently. We recommend using the control panel uninstaller to remove any previous installation(s) before installing *PGI Workstation 6.0*. If you go to the archive site on the PGI downloads page, look for the *PGI Workstation 5.2-4* release for Win32 that is described as *PGI Workstation 6.0* compatible, and re-install that version if you need to use 5.2 and 6.0 concurrently on a single system.

2.8 Customizing the Command Window

By default, when you double-left-click on the *PGI Workstation* desktop icon, a standard black-background command window appears on your screen pre-initialized with environment and path settings for use of the *PGI Workstation* compilers and tools. If you prefer different background or text colors, font style, window size, or scrolling capability, you can customize the "shortcut" that creates the *PGI Workstation* command window. Right-click on the *PGI Workstation* desktop icon, and left-click "Properties" from the pop-up menu. Modify the features mentioned above by selecting the appropriate tabs in the pop-up window and making modifications as desired.

3 PGI Release 6.0 Release Notes

This document describes changes between Release 6.0 of the PGI compilers and previous releases, as well as late-breaking information not included in the current printing of the *PGI User's Guide*. There are two versions of the PGI compilers and tools:

- A *32-bit* version supported on *32-bit operating systems* running on either a 32-bit x86 compatible or a 64-bit AMD64 or EM64T compatible processor

- A *64-bit/32-bit* version that includes all features and capabilities of the 32-bit version, and which is also supported on *64-bit operating systems* running on 64-bit AMD64 or EM64T compatible processors.

These versions are distinguished in these release notes where necessary.

3.1 PGI Release 6.0 Contents

Release 6.0 of *PGI Workstation* and *PGI Server* are comprised of the following components

- *PGF95* native OpenMP and auto-parallelizing Fortran 95

compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.

- *PGF77* native OpenMP and auto-parallelizing FORTRAN 77 compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* development environments.
- *PGHPF* data parallel High Performance Fortran compiler, in versions that will run and produce code for execution in *linux86*, *linux86-64*, and *win32* 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*, *linux86-64*, and *win32* 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. **NOTE: *PGC++ is not supported in win32 environments.***
- *PGPROF* multi-thread graphical profiler for *linux86*, *linux86-64*, and *win32* environments.
- *PGDBG* multi-thread graphical debugger for *linux86* and *linux86-64* development environments. **NOTE: *PGDBG is not supported in win32 environments.***
- Complete online documentation in PDF, HTML and UNIX `man` page formats.
- A UNIX-like shell environment for *win32* environments.

Depending on the product you purchased, you may not have licensed all of the above components.

3.2 Supported Systems

3.2.1 Supported Processors

Release 6.0 of the PGI compilers and tools is supported on the following processors. The *–tp <target>* command-line option is used to generate executables that utilize features and optimizations specific to a given CPU and operating system environment. Compilers included in a 64-bit/32-bit PGI installation can produce executables targeted to any 64-bit or 32-bit target, including cross-targeting for AMD and Intel 64-bit AMD64 compatible CPUs. CPUs available and supported in *Dual-core* versions are noted here as well.

Processors Supported by PGI 6.0-5								
Brand	CPU	Dual Core?	*<target>*	Memory Address	Floating Point HW			
					x87	SSE1	SSE2	SSE3
AMD	Opteron/Athlon64	Yes	k8-64	64-bit	Yes	Yes	Yes	Yes
AMD	Opteron/Athlon64	Yes	k8-32	32-bit	Yes	Yes	Yes	Yes
Intel	Xeon EM64T	Yes	p7-64	64-bit	Yes	Yes	Yes	Yes
Intel	Xeon EM64T	Yes	p7	32-bit	Yes	Yes	Yes	Yes
Intel	Xeon/Pentium4	No	p7	32-bit	Yes	Yes	Yes	Yes
AMD	Athlon XP/MP	No	athlonxp	32-bit	Yes	Yes	No	No
Intel	Pentium III	No	piii	32-bit	Yes	Yes	No	No
AMD	Athlon	No	athlon	32-bit	Yes	No	No	No
AMD	K6	No	k6	32-bit	Yes	No	No	No
Intel	Pentium II	No	p6	32-bit	Yes	No	No	No
Other	Other x86	No	p5 or px	32-bit	Yes	No	No	No

NOTE: Intel EM64T processors and newer-generation AMD64 processors support new floating-point hardware instructions known as SSE3. The PGI compilers make only limited use of these instructions. SSE3 instructions

will be fully supported for use in complex arithmetic in a future release of the PGI compilers and tools.

3.2.2 Supported Operating Systems

Release 6.0 of the PGI compilers and tools is supported on the operating systems listed in the table below, and their equivalents. To determine if Release 6.0 will install and run under a Linux equivalent version (Mandrake*, Debian*, Gentoo*, etc), look to see if a supported system with the same *glibc* and *gcc* versions is in the table. Version differences in other operating system components can cause difficulties, but often these can be overcome with minor adjustments to the PGI software installation or operating system environment.

Newer distributions of the Linux operating system include support for 64-bit AMD64 and EM64T compatible processors (AMD Athlon64, AMD Opteron, Intel Xeon EM64T), and are designated *64-bit* in the table. These are the only distributions on which the 64-bit/32-bit version of the PGI compilers and tools will fully install. If you attempt to install the 64-bit/32-bit version on a system running a 32-bit Linux distribution, only the 32-bit PGI compilers and tools will be installed.

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

Multi-processor AMD Opteron processor-based servers use a *NUMA* (Non-Uniform Memory Access) architecture in which the memory latency from a given processor to a given portion of memory can vary. Newer Linux distributions including SuSE 9.2/9.3 and SLES 9 include NUMA libraries that can be leveraged by a compiler and associated runtime libraries to optimize placement of data in memory.

Some older Linux distributions that were supported in the 5.2 release are not supported in release 6.0. New Linux versions like Fedora Core 3, Fedora Core 4, and RHEL 4.0 have been added as supported versions.

In the table headings, HT = *hyper-threading*, NPTL = *Native POSIX Threads Library*, and NUMA = *Non-Uniform Memory Access*. See the definitions in the Introduction to these release notes for more information on these terms.

Operating Systems and Features Supported in PGI 6.0-5									
Distribution	**Type**	**64-bit**	**HT**	**pgCC**	**pgdbg**	**NPTL**	**NUMA**	**glibc**	**GCC**
RHEL 4.0	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.4	3.4.3
Fedora C-4	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.5	4.0.0
Fedora C-3	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.3	3.4.2
Fedora C-2	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.3	3.3.3
SuSE 9.3	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.3.4	3.3.5
SuSE 9.2	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.3.3	3.3.4
SLES 9	***Linux***	Yes	Yes	Yes	Yes	No	Yes	2.3.3	3.3.3
SuSE 9.1	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.3	3.3.3
RHEL 3.0	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.2	3.2.3
SuSE 9.0	***Linux***	Yes	Yes	Yes	Yes	No	No	2.3.2	3.3.1
SuSE 8.2	***Linux***	Yes	Yes	Yes	Yes	No	No	2.3.2	3.3
Red Hat 9.0	***Linux***	No	No	Yes	Yes	Yes	No	2.3.2	3.2.2
Red Hat 8.0	***Linux***	No	No	Yes	Yes	No	No	2.2.93	3.2
SLES8 SP2	***Linux***	Poor	Yes	Yes	Yes	No	No	2.2.5	3.2.2
SuSE 8.1	***Linux***	Poor	Yes	Yes	Yes	No	No	2.2.5	3.2.2
SuSE 8.0	***Linux***	No	No	Yes	Yes	No	No	2.2.5	2.96
Red Hat 7.3	***Linux***	No	No	Yes	Yes	No	No	2.2.5	2.96
SuSE 7.3	***Will not be supported in future releases***								
SuSE 7.2	***Will not be supported in future releases***								
SuSE 7.1	***Will not be supported in future releases***								
Red Hat 7.2	***Will not be supported in future releases***								
Red Hat 7.1	***Will not be supported in future releases***								
Red Hat 7.0	***Will not be supported in future releases***								
Microsoft Windions	*XP*	No	Yes	No	No	NA	NA	NA	NA
	2003	No	No	No	No	NA	NA	NA	NA
	2000	No	No	No	No	NA	NA	NA	NA

NOTE: *http://www.pgroup.com/support/install.htm* lists any new Linux distributions that may be explicitly supported by the PGI compilers. If your Linux distribution is newer than any of those listed in the table above, the installation may still be successful.

3.3 New or Modified Compiler Features

Following are the new features of Release 6.0 of the PGI compilers and tools as compared to prior releases.

- *Fortran 95* – the *PGF95* compiler supports full Fortran 95. The command name `pgf95` is added, but `pgf90` is retained to enable full backward compatibility of build environments with previous releases of the PGI compilers and tools. In addition, the *.f95* and *.F95* input file suffixes are now recognized.
- *C/C++ template instantiation* – the *PGC++* compiler now supports GNU-style template instantiation. This new method uses the GNU linker to resolve all template references and discard the duplicates, greatly simplifying the use of templates.
- *Large Arrays* – single data objects larger than 2GB in size are now supported by all PGI Fortran, C and C++ compilers and tools in *linux86-64* environments. Data objects, both locally and globally declared, including objects in COMMON blocks and objects allocated on the stack, can be larger than 2GB individually and in aggregate. The *–mcmodel=medium* option must be used to compile programs that use *Large Arrays*. The option *–Mlarge_arrays* is available and can be used for certain applications designed to run in a small memory model with a single large data space allocated and managed by the application.
- *64-bit efficiency* – improved optimization of loops and code segments operating on large arrays results in improved

efficiency of 64-bit programs.

- *Auto-parallel/OpenMP efficiency* – improved optimization and vectorization of OpenMP or auto-parallelized parallel loops results in improved efficiency of OpenMP programs for both single-processor and multi-processor execution. In addition, process-to-CPU scheduling affinity capabilities are enabled for operating systems that support this feature, such as SuSE 9.2.
- *Performance* – Further tuning of the 32-bit x86 and 64-bit AMD64 and EM64T intrinsic libraries, code generators and other optimization phases have resulted in performance improvements averaging 5% to 15% over the previous PGI 5.2 release as measured by several industry-standard benchmark suites and applications.
- *C/C++ Performance* – Several optimizations specific to C/C++, including support of ANSI pointer semantics, improved pointer disambiguation, math intrinsics optimizations, block placement, loop unrolling, idiom recognition, and structure optimizations have contributed to an overall improvement in C/C++ performance.
- *Position-independent static runtime libraries* – versions of the PGI runtime libraries built with *–fpic* as static archives are now available. This allows creation of statically-linked 64-bit executables.
- *Profile-feedback optimizations* – many common profile-feedback optimizations, semi-invariant value optimizations, and block placement have been added under control of the new *–Mpfi*/*–Mpfo* command-line options described in detail later in these release notes.
- *IPA enhancements* – the inter-procedural analysis (IPA) and optimization phases of the PGI compilers have been enhanced to support IPA-based optimizations to occur between program units being newly-compiled from source and those that are linked in from static archives (libraries). Several other

enhancements to IPA which contribute to overall improved performance have been implemented.

- *Prefetch directives – PGF77* and *PGF95* support directives and *PGCC* and *PGC++* support pragmas to allow explicit prefetching of data by the programmer. These directives together with the *–Mprefetch* command-line option have been enhanced to allow explicit control over the type of prefetch instructions generated by the compilers. See the *PGI User's Guide* for details on how to use these directives and pragmas.
- *RANDOM_NUMBER* – In previous releases, calls to RANDOM_SEED without arguments would result in generation of the same sequence of random numbers at each execution of a program. In Release 6.0, the first call to RANDOM_SEED without arguments will reset the random seed to a default value, then advance the seed by a variable amount based on time. Subsequent calls to RANDOM_SEED without arguments will reset the random seed to the same initial value as the first call. Unless the time is exactly the same, each time a program is run a different random number list will be generated. This change has no effect on the RANDOM_SEED PUT and GET arguments. You can force the pre-6.0 RANDOM_SEED behavior by setting the environment variable `STATIC_RANDOM_SEED` to `yes`.
- *ACML 2.5.3* – the latest edition of the *AMD Core Math Library, ACML 2.5.3*, is bundled with the PGI 6.0 compilers and tools. The bundled version of the ACML supports only 32-bit *x86* and 64-bit *AMD64*-compatible CPUs that support both SSE1 and SSE2 instructions. The lower-performance but fully portable *libblas.a* and *liblapack.a* libraries are still included, and can be used on CPUs that do not support SSE instructions. **NOTE**: ACML 2.5.3 is built using the *–fastsse* compile/link option, which includes *–Mcache_align*. When linking in the ACML 2.5.3, you must compile/link all program units with *–Mcache_align*, or an aggregate option such as *–fastsse* which

incorporates *–Mcache_align*.

- *EM64T support* – Intel IA32 processors with EM64T extensions, designed to be binary compatible with AMD64 technology processors from AMD, are supported and can be targeted for optimizations specifically using the *–tp p7-64* command-line option. Optimizations and instructions selection are targeted for EM64T when this option is used.
- *Expanded OS support* – several new Linux distributions are supported, including *SuSE 9.2*, SuSE 9.3, *SLES 9, RHEL 4.0*, *Fedora Core 3* and *Fedora Core 4*.
- *REDIST directory* – runtime libraries that are re-distributable under the terms of the PGI End-user License Agreement are collected in `/usr/pgi/linux86/6.0/REDIST`, `/usr/pgi/linux86-64/6.0/REDIST`, and, on *Win32*, C:/Program Files/PGI/nt86/6.0/REDIST.
- *Site-specific rc configuration file* – create a compiler configuration file, `$PGI/<target>/6.0/bin/siterc`, to customize defaults for the PGI compilers and tools at your site.
- *Dual-core support* – As of Release 6.0-5, the PGI compilers and tools are fully supported on Dual-core processors from AMD and Intel. See the new multi-processor (MP) environment variables in section 3.4.3 for more information.
- *NUMA support* – As of Release 6.0-5, the PGI compilers leverage the NUMA libraries included with SuSE 9.2/9.3 and SLES 9 to optimize placement of data in OpenMP programs. See the new *–mp=numa* OpenMP parallelization option in section 3.4.2 for more information.

3.4 Compiler Options

3.4.1 Getting Started

By default, the PGI 6.0 compilers generate code optimized for the type of processor on which compilation is performed (the compilation host). If you are unfamiliar with the PGI compilers and tools, a good option to use by default is *–fast*. This option is host-dependent but usually includes the options *–O2 –Munroll –Mnoframe*. Typically, for best performance on processors that support SSE instructions, you will want to use the *PGF95* 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 vector streaming SIMD (SSE/SSE2) instructions where appropriate. The contents of the *–fastsse* switch are host-dependent, but typically include the options *–O2 –Munroll –Mnoframe –Mlre –Mvect=sse –Mcache_align*. On some systems, *–fastsse* also includes *–Mscalarsse* and *–Mflushz*.

In addition to *–fastsse*, the *–Mipa=fast* option for inter-procedural analysis and optimization can improve performance. You may be able to obtain further performance improvements by experimenting with the individual *–Mpgflag* options detailed in the *PGI User's Guide* (*–Mvect*, *–Munroll*, *–Minline*, *–Mconcur*, *–Mpfi*/*–Mpfo*, 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.

3.4.2 New or Modified Compiler Options

The following compiler options have been added or modified in PGI 6.0:

- *–Mipa*[= …, *safe:*<*name*>, *safeall*, …] – instructs the compiler to assume that calls to program units in *name* are safe, even if those program units are not compiled with IPA, and should not inhibit IPA optimizations in the caller. The *name* can be either a library filename or the name of a function. Using *–Mipa=safeall*

instructs the compiler to assume that all libraries linked into the executable are safe in this regard.

- *–Mipa*[=…[,*libinline, libopt*[,…]] – Using *–Mipa=libinline* instructs the compiler to attempt to inline functions from libraries that have been compiled using *–Mipa*. Using *–Mipa=libopt* instructs the compiler to perform inter-procedural optimizations on functions contained in libraries that have been compiled using *–Mipa*.

- *–Mnoipa* – disable interprocedural analysis and optimization (IPA). Can be used on the command-line after an aggregate option to disable IPA without affecting the other options included in the aggregate option.

- *–Mnovect* – disable vectorization. Can be used on the command-line after an aggregate option such as *–fastsse* to disable vectorization without affecting the other options included in the aggregate option.

- *–M[no]daz* – (disable) enable flush-to-zero mode for IEEE 754 denormalized numbers.

- *–Mfprelaxed* – where profitable, enable a fast but less accurate algorithm for computation of single-precision floating-point divide, square root and reciprocal square root using reciprocal approximation and inline Newton's iterations.

- *–M[no]large_arrays* – (disable) enable support for single static data objects larger than 2GB in size. Applies to the *PGF95*, *PGF77*, *PGCC*, and *PGC++* compilers. This option is implied by *–mcmodel=medium*, and is required for applications that use single static data objects larger than 2GB in size.

- *–M[no]prefetch*[=*d:*<***m***>[,*n*:<**p**>[,{nta | *t0* | *w*}]]] – (disables) enables generation of prefetch instructions; only applies when used in combination with *–Mvect* or an aggregate option such as *–fastsse* that incorporates *–Mvect*. The new *d:*<***m***> distance sub-

option instructs the compiler to prefetch data a distance of ***m*** cache lines ahead of the data currently being accessed. The new *n:<**p**>* number sub-option instructs the compiler to issue up to **p** prefetch instructions in loops where prefetching is used. The new *nta*, *t0*, and *w* sub-options instruct the compiler to use `prefetchnta`, `prefetcht0` or `prefetchw` instructions for prefetching. *NOTE*: the `prefetchw` instruction is not supported on IA32 or EM64T processors.

- *–Mpfi* – instrument the generated executable for collection of profile and data feedback information. This information can be used in subsequent compilations that include the *–Mpfo* optimization option. Programs compiled with *–Mpfi* will execute more slowly due to this instrumentation and data collection overhead.

- *–Mpfo* – use data from a `pgfi.out` profile feedback tracefile to enable or enhance certain performance optimizations.

- *–Mprof[=option,option,...]* – Set profile options. Normally, the *–ql*, *–qp*, or *–pg* switches are used for profiling; however, on some systems, it is desirable to override the default method of profiling. See the *PGI User's Guide*, or the system profiler manual, for further information:

dwarf	generate limited DWARF information to enable source correlation by 3rd-party profiling tools.
func	Perform PGI-style (instrumented) function-level profiling.
hwcts	Use PAPI-based profiling with hardware counters (*linux86-64* platforms only).
lines	Perform PGI-style (instrumented) line-level profiling.
mpi	Use and link MPICH-style profiling (*PGI CDK* only).
time	Sample-based instruction-level profiling.

- *–Mvarargs* – Force Fortran program units to assume calls are to C functions with a `varargs` type interface.

- *–mcmodel=medium* – In Release 6.0 of the PGI compilers and tools, *–mcmodel=medium* now implies *–Mlarge_arrays*. This means you typically can create full 64-bit executables by simply adding *–mcmodel=medium* to compilation/linking of all files in a program.

- *–mp*[=[*no*]*align* | *numa*] – The *–mp* option enables recognition and processing of OpenMP directives and pragmas. The new *align* sub-option forces loop iterations to be allocated to OpenMP processes using an algorithm that maximizes alignment of vector sub-sections in loops that are both parallelized and vectorized for SSE/SSE2. This can improve performance in program units that include many such loops. It can result in load-balancing problems that significantly decrease performance in program units with relatively short loops that contain a large amount of work in each iteration. The new *numa* sub-option is supported only on AMD64 processors running a suitable version of Linux, and includes optimizations specific to *Non-Uniform Memory Access* (NUMA) systems where the memory latency is dependent on the location of data in the system relative to a given processor. ***NOTE*:** the *–mp=numa* option is only valid on SuSE 9.2/9.3 and SLES 9.

- *–pgf77libs* – Link in *PGF77* runtime libraries; for use with the `pgcc` or `pgCC` drivers when linking *PGF77*-compiled object files into *C* or *C++* main programs.

- *–pgf90libs* – Link in *PGF95* runtime libraries; for use with the `pgf77`, `pgcc` or `pgCC` drivers when linking PGF95-compiled object files into F77, C or C++ main programs.

3.4.3 New or Modified Environment Variables

Three environment variables are available to control the behavior of threads or processes used to execute parallel regions in OpenMP or auto-parallel programs for SMP or Dual-core processor-based systems:

MP_BIND, MP_BLIST and MP_SPIN. Using these variables, you can control whether or not threads can migrate from one processor to another, how threads are allocated to processors, and how aggressively idle threads will consume processor cycles while waiting at a barrier.

- MP_BIND - the MP_BIND environment variable can be set to yes or y to bind processes or threads executing in a parallel region to physical processors, or to no or n to disable such binding. If this environment variable is not set, the default is to *not* bind processes to processors. This is an *execution time* environment variable interpreted by the PGI runtime support libraries. It does not affect the behavior of the PGI compilers in any way. It is only advisable to use this environment variable when running programs on a standalone system, as a means to prevent spurious remapping of processes by the operating system. ***NOTE*:** the MP_BIND environment variable is only supported on SuSE 9.2/9.3 and SLES 9, and only for programs that are compiled using *–mp=numa*.

- MP_BLIST - In addition to the MP_BIND variable, it is possible to define the thread-CPU relationship. For example, setting MP_BLIST=3,2,1,0 maps CPUs 3, 2, 1 and 0 to threads 0, 1, 2 and 3 respectively. ***NOTE*:** the MP_BLIST environment variable is only supported on SuSE 9.2/9.3 and SLES 9, only for programs that are compiled using *–mp=numa*, and only has an effect if MP_BIND is set to yes or y.

- MP_SPIN - When a thread executing in a parallel region enters a barrier, it spins on a semaphore. MP_SPIN can be used to specify the number of times it checks the semaphore before calling sched_yield() (on linux) or _sleep() (on Win32). These calls cause the thread to be re-scheduled, allowing other processes to run. The default values are 1000000 (Linux) and 100000 (Win32). If MP_SPIN is set to -1, the threads will spin on the semaphore indefinitely without calling sched_yield().

In addition to these environment variables, the numactl command available on SuSE 9.2/9.3 and SLES 9 can be very useful as a means of controlling memory layout and allocation on NUMA AMD64 processor-based systems. See the man page for numactl for more information.

To monitor the assignment of processes to processors, the `xosview` utility can be very useful. See the man page for `xosview` for more information.

3.4.4 C++ Template Instantiation Changes

C++ template instantiation has changed for 32-bit and 64 bit Linux target systems. The new method uses the GNU linker to resolve all template references and discard duplicates. This new feature is not compatible with previous releases of the PGI compilers and tools. To migrate C++ codes to *PGC++ 6.0*, it is necessary to recompile all C++ code, and modify makefiles to remove all template instantiation flags. The following template-related command-line switches have been deprecated:

—one_instantiation_per_object

—instantiation_dir

—instantiate

—[no_]auto_instantiation

—prelink_objects

–Wc, –tlocal

–Wc, –tused

–Wc, –tall

3.4.5 The REDIST Directory

The PGI 6.0 release includes directories named *$PGI/linux86/6.0/REDIST* and *$PGI/linux86-64/6.0/REDIST*, and *$PGI/nt86/6.0/REDIST* on *Win32* These directories contain all of the PGI runtime library shared object files or, on *Win32*, dynamically linked libraries that can be re-distributed by PGI 6.0 licensees under the terms of the PGI End-user License Agreement (ELA), a copy of which is included in these directories in text form for reference.

The REDIST directories contain the PGI runtime library shared objects for

all supported targets. This enables users of the PGI compilers to create packages of executables and PGI runtime libraries that will execute successfully on almost any PGI-supported target system, subject to the requirement that end-users of the executable have properly initialized their environment and LD_LIBRARY_PATH to use the relevant version of the PGI shared objects.

3.4.6 Customizing With *siterc* and User *rc* Files

The PGI 6.0 release includes a `siterc` file in the `bin` directory to enable site-specific customization of the PGI compiler drivers. Using `siterc`, you can control how the compiler drivers invoke the various components in the compilation tool chain. In addition to the `siterc` file, user *rc* files can reside in a given user's home directory: `.mypgf77rc`, `.mypgf90rc`, `.mypgccrc`, `.pypgCCrc`, and `.mypghpfrc` can be used to control the respective PGI compilers. All of these files are optional.

Following are some examples with show how these *rc* files can be used to tailor a given installation for a particular purpose.

Make the libraries found in */opt/newlibs/64* available to all *linux86-64* compilations	Add the line: `set SITELIB=/opt/newlibs/64;` to */usr/pgi/linux86-64/6.0/bin/siterc*
Make the libraries found in */opt/newlibs/32* available to all *linux86* compilations.	Add the line: `set SITELIB=/opt/newlibs/32;` to */usr/pgi/linux86/6.0/bin/siterc*

Add a new library path `/usr/local/fast` to *all linux86-64 compilations.*	add the line: `append SITELIB=/usr/local/fast;` to */usr/pgi/linux86-64/6.0/bin/siterc*
Make the include path `/opt/acml/include` available to all compilations; *–I/opt/acml/include*.	add the line: `set SITEINC=/opt/acml/include;` to */usr/pgi/linux86/6.0/bin/siterc* and */usr/pgi/linux86-64/6.0/bin/siterc*

Change *–Mmpi* to link in */opt/mympi/64/libmpix.a* with *linux86-64* compilations.	add the the lines:
	`set MPILIBDIR=/opt/mympi/64;`
	`set MPILIBNAME=mpix;`
	to */usr/pgi/linux86-64/6.0/bin/siterc*

Have *linux86-64* compilations always add *–DIS64BIT –DAMD*	add the line: `set SITEDEF=IS64BIT AMD;` to */usr/pgi/linux86-64/6.0/bin/siterc*
A user wishes to build an F90 executable for *linux86-64* or *linux86* that resolves PGI shared objects in the relative directory *./REDIST*	add the line: `set RPATH=./REDIST;` to *~/.mypgf95rc*. *NOTE:* this will only affect the behavior of PGF95 for the given user

3.5 64-bit Support

The *PGF77* and *PGF95* compilers included in Release 6.0 support both the

–mcmodel=small and *–mcmodel=medium* addressing models as defined in the *X86-64 Application Binary Interface*. The following table summarizes the limits of these programming models.

Programming Models on 64-bit Linux86-64 Systems						
Combined Compiler Options	**Addr. Math**		**Max Size Gbytes**			**Comments**
	A	**I**	**AS**	**DS**	**TS**	
–tp k8-32 or –tp p7	32	32	2	2	2	32-bit *linux86* programs
–tp k8-64 *or* –tp p7-64	64	32	2	2	2	64-bit addr, limited by *–mcmodel=small*
–tp k8-64 –fpic *or* –tp p7-64 –fpic	64	32	2	2	2	*–fpic* incompatible with *–mcmodel=medium*
–tp k8-64 or –tp p7-64 –mcmodel=medium	64	64	>2	>2	>2	Enable full support for 64-bit data addressing

Column Legend	
A	*Address Type* (**A**) -size in bits of data used for address calculations, 32-bit or 64-bit.
I	*Index Arithmetic* (**I**)- bit-size of data used to index into arrays and other aggregate data structures. If 32-bit, total range of any single data object is limited to 2GB.
AS	*Maximum Array Size* (**AS**)- the maximum size in bytes of any single data object.
DS	*Maximum Data Size* (**DS**)- max size in bytes combined of all data objects in *.bss*
TS	*Maximum Total Size* (**TS**) max size in bytes, in aggregate, of all executable code and data objects in a running program.

The program area is the total area used by the Linux operating system and the user program. On most 32-bit Linux systems, only about 1GB is available for data (in theory 2GB is accessible with a 32-bit signed integer address).

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

Support for the *medium memory model* in the *linux86-64* environment is provided using the *–mcmodel=medium* compile and link option. The *medium memory model* allows for larger than 2GB data objects and *.bss* sections. Object files linked into an executable requiring the *–mcmodel=medium* link-time option must be compiled using either *–mcmodel=medium* or *–fpic*, but cannot be compiled using both of these options.

3.5.1 Practical Limitations of –mcmodel=medium

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

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

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

page swapping — 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.

configured space

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, to determine if a program failure is due to a compiler or an operating system limitation.

3.5.2 Large Array Example in C

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

```
% cat bigadd.c
```

```
#include <stdio.h>
#define SIZE 600000000  /*  > 2GB/4 */
static float a[SIZE],b[SIZE];
main() {
long long i,n,m;
float c[SIZE];  /* goes on stack */
n=SIZE;m=0;

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

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

```
% gcc –o bigadd bigadd.c
```

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

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

```
% gcc –mcmodel=medium –o bigadd bigadd.c
/tmp/ccVQpbPj.s: Assembler messages:
/tmp/ccVQpbPj.s:97: Error: .COMMon length (-2147483648.)
<0! Ignored.
```

The *gcc* compiler incorrectly converts a greater than 2G value to a *negative* 32-bit number in an assembler statement. This error does not occur using *pgcc 6.0*:

```
% 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 6.0 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
```

3.5.3 Large Array Example in Fortran

The following example works with both the *PGF95* and *PGF77* compilers included in Release 6.0. Both compilers use 64-bit addresses and index arithmetic when the *–mcmodel=medium* option is used.

Consider the following example:

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

      do i = 1, m
         do j = 1, n
            a(I,j)=10000.0D0*dble(i)+dble(j)
            b(I,j)=20000.0D0*dble(i)+dble(j)
         enddo
      enddo
!$omp parallel
!$omp do
      do i = 1, m
         do j = 1, n
            c(i,j) = a(i,j) + b(i,j)
         enddo
      enddo
!$omp do
```

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

When compiled with the *PGF95* compiler using *–mcmodel=medium*:

```
% pgf95 –mp –o mat mat.f –mcmodel=medium

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

3.5.4 Large Array Example in C++

The example below compiles and executes correctly when compiled with *PGC++* using the switch *–mcmodel=medium*. Please note that large classes that are allocated dynamically will cause execution failure if the process stack is not large enough. In this case, set the process stack to an appropriate limit in your environment:

```
% limit stacksize 12000M
% cat example.c
#include <iostream>
using namespace std;
#define SIZE 400000000 /* > 2GB/2 */
class large{
```

```
public:
        double sum();
        large () {
               int i;
               for(i=1; i<=SIZE; i++) {
                      big1[i] = 1.0;
                      big2[i] = .5;
               }
        }

private:
        double summer;
        float  big1 [SIZE];
        float  big2 [SIZE];
};
double large::sum()
{
         int i;
         summer=0.0;
         {
                  for(i=1; i<=SIZE; i++) {
                        summer +=  .00000001 * big1[i] +
                                   .00000001 * big2[i] ;
                  }
         }
         return summer;

};
large total;
int main() {
         double value;
         value=total.sum();

         cout << "total = " << value  << endl;
};
```

When compiled with the *PGC++* compiler using *–mcmodel=medium*:

```
% pgCC -fast –mcmodel=medium example.c
% a.out
total = 18
```

3.6 PGI Workstation 6.0 for Win32

PGI Workstation 6.0 for *Win32* environments has most of the features of the 32-bit version for *linux86* environments, with the following main differences:

- The *PGC++* compiler is not available for *Win32* environments
- The *PGDBG* debugger is not available for Win32 environments
- The *PGPROF* performance profiler is included, but only with a command-level text interface (no graphical user interface)

The product provides a familiar and somewhat compatible development environment for Linux or RISC/UNIX users porting or developing programs for the 32-bit Windows Operating system. However, there are some *C* library routines specific to Unix/Linux that are not provided in the *MinGW32* UNIX-like command environment included with *PGI Workstation 6.0* for *Win32*.

On *Win32*, a UNIX-like shell environment is bundled with *PGI Workstation 6.0*. After installation, a double-left-click on the *PGI Workstation* icon on your desktop will launch a *bash* shell command window with pre-initialized environment settings. Many familiar UNIX commands are available (`vi`, `emacs`, `sed`, `grep`, `awk`, `make`, etc). If you are unfamiliar with the *bash* shell, refer to the user's guide included with the online HTML documentation.

Alternatively, you can launch a standard *Win32* command window pre-initialized to enable use of the PGI compilers and tools by selecting the appropriate option from the *PGI Workstation* program group accessed in the usual way through the "Start" menu.

Except where noted in the *PGI User's Guide*, the command-level compilers and tools on *Win32* function identically to their *Linux* counterparts. You can customize your command window (white background with black text, add a scroll bar, etc.) by right-clicking on the top border of the *PGI Workstation* command window, selecting "Properties", and making the

appropriate modifications. When the changes are complete, *Win32* will allow you to apply the modifications globally to any command window launched using the *PGI Workstation* desktop icon.

3.7 PGDBG and PGPROF

PGDBG is supported as a graphical and command line debugger in both the *linux86* and *linux86-64* execution and development environments. Like the compilers, *PGDBG* for *linux86-64* must run in a *linux86-64* execution environment. *PGDBG* for *linux86* environments is a separate version, and it will also run in the *linux86-64* execution environment, but only with *linux86* executables. The *linux86-64* version of *PGDBG* will only debug executables built to run as *linux86-64* executables. *PGDBG* for *linux86-64* has been enhanced to disassemble *AMD64* and *EM64T* technology instructions and associated registers, and is more compatible with *gcc*, *g77*, and *g++* debug information.

PGPROF is supported as a graphical and command line profiler in both the *linux86* and *linux86-64* environments. The same version works in either the *linux86 or linux86-64* environment to process a trace file of profile data created by executing the instrumented program. Program instrumentation is either line-level (*–Mprof=lines*), function-level (*–Mprof=func*), or *gprof*-style (*–pg*) sample based and trace profiling.

The *PGDBG* and *PGPROF* graphical user interfaces (GUIs) are invoked by default. To use a command line interface, invoke either tool with the *–text* option. The motif GUI interfaces provided in *PGI Workstation 5.1* and prior releases are no longer supported.

3.7.1 PGDBG and PGPROF New Features

PGI Workstation 6.0 includes several new features and enhancements in the *PGDBG* parallel debugger and *PGPROF* performance profiling tools.

- *Core files – PGDBG* is now able to debug core files on *linux86* and *linux86-64* systems.

- *Array displays – PGDBG* now prints the contents of arrays in a user-friendly format showing subscript information. Array slices are supported in this way as well.

- *Enhanced Fortran 95 debugging support – PGDBG* has been enhanced with numerous minor improvements related to support for F95 modules and derived data types. For example, *PGDBG* now supports the % character as a field accessor for derived types, and supports better symbolic access to many F90/F95 data items.

- *Viewing OpenMP private data – PGDBG* is now able to accurately present the contents of all OpenMP PRIVATE data items.

- *MPI processes cleanup – PGDBG* now completely cleans up MPI listener processes upon exit.

- *Enhanced Thread Support – PGDBG* requirements for the user to set environment variables or link in certain ways for threads debugging have been removed. *PGDBG* now supports debugging OpenMP and multithreaded programs without any restrictions on the system configuration, environment, or the way the application was linked.

- *Standard "run" command behavior –* the *PGDBG* `run` command must now be the first control command executed in a debug session (before `step`, `cont`, and so on). This should prevent user confusion caused by stepping into the program startup code on some operating systems.

- *Automatic display of SSE registers – PGDBG* now automatically detects whether it is running on an SSE-capable processor, and if so dumps SSE register contents by default as part of the `regs` command.

- *Hardware counter-based performance profiling – PGPROF* now supports low-overhead hardware counters-based profiling for 64-bit Linux using the Performance Application Programming Interface (PAPI), available at *http://icl.cs.utk.edu/papi*. To use

this feature, your system must be running a version of the Linux operating system patched using the *PerfCtr* Linux kernel patch, which is included in the PAPI distribution.

- *Assembly-level performance profiling – PGPROF* supports performance profiling at the function, block and now the assembly line level. Source and assembly code can be interleaved when viewing profiles. This feature is only available in profiles using hardware performance counters or *gprof*-style sample-based profiling.

- *Profile visualization through graphical histograms – PGPROF* now supports a histogram window for more compact and intuitive analysis of the performance of program units.

- *Profile search mechanism* – The *PGPROF GUI* now supports a search menu with various options for searching through profile information.

- *Profiler command-line interface – PGPROF* now includes an all-new command-line (non-graphical) interface that supports all features included in the GUI version.

- *Sample-based MPI profiling – PGPROF* now supports lower-overhead sample-based profiling for MPI programs.

- *Improved scalability comparisons – PGPROF* includes new formulas for computing scalability. Refer to section 2.2.5 of the *PGI Tools Guide* for details on this feature.

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

See the *PGI Tools Guide*, completely updated for *PGI Workstation 6.0*, for a complete description of the usage and capabilities of *PGDBG* and *PGPROF*. For tool limitations and workarounds, see the FAQ *http://www.pgroup.com/support/faq.htm*.

3.8 Known Limitations

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

- Object and module files created using *PGI Workstation 6.0* compilers are incompatible with object files from *PGI Workstation 5.X* and prior releases.

- The *–i8* option can make programs incompatible with MPI and the ACML math library. Typically, use of any INTEGER*8 array size argument can cause failures with these libraries.

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

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

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

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

- *OpenMP* programs compiled using *–mp* and run on multiple

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

- ACML 2.5 is built using the *–fastsse* compile/link option, which includes *–Mcache_align*. When linking in the ACML 2.5 using the *–lacml* option on 32-bit targets, you must compile/link all program units with *–Mcache_align*, or an aggregate option such as *–fastsse* which incorporates *–Mcache_align*. This is not an issue on 64-bit targets where the stack is 16-byte aligned by default.

- Times reported for multi-threaded sample-based profiles (profiling invoked with *–pg* or *–Mprof=time* options) are for the master thread only. PGI-style instrumentation profiling with *–Mprof=*{*lines* | *func*} or hardware counter-based profiling using *–Mprof=hwcts* must be used to obtain profile data on individual threads.

- *PGDBG* GUI – from command pane, the `source` command does not wait for execution to stop. It continues to read commands, even if the target is running. For example, if the `source` script contains commands to set a breakpoint, run and print a stack trace, the expectation might be that the stack trace would print at the breakpoint. In fact, it might return an error, since the target could be running when `stacktrace` is executed. The only workaround is to insert a `wait` command after each control command in the script.

- *PGDBG* – the `watch` family of commands is unreliable when used with local variables. Calling a function or subroutine from within the scope of the `watch`ed local variable may cause missed events and/or false positive events. Local variables may be `watch`ed reliably if program scope does not leave the scope of the `watch`ed variable. Using the `watch` family of commands with global or static variables is reliable.

- *PGDBG* – the `stacktrace` command can execute very slowly. The method currently used by *PGDBG* to retrieve function or subroutine arguments in the `stacktrace` command is not at all optimal. As a temporary workaround, the undocumented `calltrace` command can be used for faster tracebacks without argument information; `calltrace` may not be supported in future releases.

- *PGDBG* – the `stacktrace` command may skip a frame in the call stack if it encounters a routine compiled without *–g*. This is most noticeable when an exception is encountered in a library routine such as `memset()`, and `stacktrace` does not show the calling routine. The current routine may not be identified by name, showing only `unknownaddr`. There is no known workaround for this problem.

- *PGDBG* – issuing a `step` command to step into a function or subroutine in a shared library compiled *–g* fails; the command steps over the function. As a workaround, set a breakpoint on the function name and use the `cont` command to continue to the breakpoint in the shared library.

- *PGDBG* – the `call` command does not support the following F90/F95 features: array-valued functions, pointer-valued functions, assumed-shape array arguments, pointer arguments. There is no known workaround to this limitation.

- *PGDBG* – if you execute a `run` or `rerun` command with no arguments after a previous `run` or `rerun` that specified I/O re-direction, I/O redirection continues as specified in the previous `run` or `rerun`. This can cause unexpected results to be appended to a file specified as *stdout*, and can cause unexpected program failures due to erroneous program input from *stdin* which is not reset to the start of the intended input file. This limitation also applies to a `shell` command issued after a `run` or `rerun` with I/O redirection.

3.9 Corrections

The following problems have been corrected in the *PGI Workstation 6.0* release. Most were reported in *PGI Workstation 5.2* or previous releases. Problems found in *PGI Workstation 5.2* may not have occurred in the previous releases. A table is provided that describes the summary description of the problem. An *Internal Compiler Error* (ICE) is usually the result of checks the compiler components make on internal data structures, discovering inconsistencies that could lead to faulty code generation. For a complete and up-to-date list of TPRs fixed in recent releases of the PGI compilers and tools, see *http://www.pgroup.com/support/release_tprs.htm*.

TPR	Rel	Lang/ Tool	Description
2353	3.1	pgdbg	Inability to access F90/F95 deferred shape arrays
2979	3.1	pgdbg	Inability to debug app's linked with certain thread lib's
3027	5.0	pghpf	program causes pghpf,pgf90 ICE
3050	5.0	pgf90	pgf90 private variables not acting private
3119	5.1	pgf90	pgf90 private variables not acting private
3140			Closed Invalid
3150			Closed Invalid
3195	5.1	pgf90/f77	Provide better IOSTAT information
3206	5.1	pgCC	C++ MAXIDLEN too small for multiply nested templates
3208	5.1	win32	win32 does not install with other compilers
3228	5.2	pgf90	non-terminating string caused pgf90 to coredump
3244	5.1	pgdbg	supports core files
3250	5.1	pgf90	pgf90 program causes ICE
3256	5.2	pgf90	alignment errors with data-initialized COMMONs
3257	5.1	pgf90	random_seed does not change random_number
3258	5.2	all	possible CG problem 32/64-bit compilers.
3259	5.2	pgf90	pgf90 character array constructors not properly handled
3261	5.2	all	32/64 bit designation with -V fixed in 5.2
3264	5.2	pgf90/f77	large file problems with 64-bit programs
3269	5.2	pgcc	-fastsse causes ICE flowgraph: node is zero - signal 11
3270	5.2	pgf90	pgf90 ICE Lowering Error:
3271	5.2	pgcc	pgcc program causes ICE and signal 11

TPR	Rel	Lang/ Tool	Description
3273	5.2	pgcc	long long test failures
3275	5.2	all	16-byte alignment to conform to x86-64 ABI
3297	5.2	pgf90	debug information correction
3304	5.2	pgCC	pgCC does not have large file support
3305	5.2	pgf90/f77	fortran ‘internal file’ problem
3310	5.2	pgCC	OpenMP parallel reduction fails to compile with pgCC
3313	5.2	pgf90	program generates inefficient code
3314	5.2	pgf90	MAXLOC not following pgf90 standard
3315	5.2	pgf90	adjustl and trim together in pgf90 fail
3318	5.2	pgf90	pgf90 Non-constant expression severe error
3321	5.2		Closed - invalid
3322	5.2	pgf90	pgf90 programs generate Argument type mismatch
3324	5.2	pgf90	cpu_time does not work as real, only as real*8
3326	5.2	pgCC	Request for New style C++ template instantiation
3328	5.2	pgf90	pgf90 handling C varargs functions –Mvarargs created
3329	5.2	pgf90	same as 3257
3330	5.2	openmp	Added the include guard _PGOMP_H to omp.h
3331	5.1	pgf90	array-valued adjustl/adjustr incorrect
3333	5.2	pgf90	pgf90 program gives wrong answers
3334	5.2	pgf90	program causes ICE Errors in Lowering 3
3347	5.2	pgf90	undefined reference to `pghpfio_byte_read'
3353	5.2	pgf90	MOM4 build causes pgf90 ICE
3362	5.0	pgf90	F90 ICE for derived type constructor
3366	5.2	pgf90	MOM4 -- pgf90 quoted string syntax error
3367	5.2	pgf90	MOM4 -- Non-comformable array objects
3369	5.2	pgf90	pgf90 loop counters not affected by -i8
3370	5.2	pgcc	-O2 causes seg fault, 64-bit compiles only
3371	5.2	pgf90	pgf90 ICE IM_BASE op#2 not based sym
3372	5.2	pgCC	openmp support in C++ does not include template indexes
3373	5.2	all	drivers not properly including -lpgsse1 -lpgsse2
3374	5.2	pgf90	pgf90 -mp ICE ' Errors in Lowering'
3376	5.2	pgf90	pgf90 program causes ICE - _dinit_acl,array
3377	5.2	pgf90/f77	fortran program fails -Ktrap=fp at -O2
3382	5.2	pgcc	pgcc preprocessor fails with correct syntax
3383	5.2	pgf90	pgf90 program gives wrong answers with 64-bit -O1
3384	5.2	pgf90	Problem with nested where statements in f90
3385	5.2	pgf90	pgf90 false allocation detection

TPR	Rel	Lang/ Tool	Description
3387	5.2	pgf90	pgf90 compilation terminates with signall 11 at -fastsse
3388	5.2	pgf90	pgf90 object is extremely large -- init'd module variables
3389	5.2	pgf90	pgf90 program causes ICE - Errors in Lowering
3390	5.2	pgCC	C++ large objects take up too much space in.o
3391	5.2	pgdbg	Debugger problem running PGI Workstation
3392	5.2	pgf90	ICE assem.c-put_skip old,new not in sync in GRIB2
3393	5.2	pgf90	pgf90 code causes 'pgf901 TERMINATED by signal 11'
3394	5.2	pgf90/f77	linux86-64 has problems with record size >2GB
3395			Closed Invalid
3399	5.2	pgf90/f77	linking -g with include files fails
3401	5.2	pgf90	pgf90 ICE: cgasm_oprnd: cant assmbl opernd 29
3402	5.2	pgf90	ICE: Errors in ILM file
3404	5.2	pgf90/f77	formatted output does not print out very large arrays
3405			Closed Invalid
3406			Closed Invalid
3409	5.2	pgf90	64-bit pgf902 seg fault with "-g"
3410	5.2	pgf90	pgf90 terminates from too many initializations
3412	5.2	pgf90	pgf90 reject legal code 'type mismatch'
3413	5.2	pgf90/f77	pgf90 does not properly handle nan as said in standard
3414	5.2	pgf90	pgf90 program, compiled -tp k8-64 -fastsse hangs
3415	5.2	pgf90	Polyhedron Induct compiles with -Mipa=fast,inline -Munroll=n:4 too long
3417	5.2	pgf90	program causes occasional signal 11
3418	5.2	pgf90	program executes different from other compilers
3419			Closed Invalid
3420	5.2	pgf90	program gives wrong answers on tp k8-64 optimizations
3422	5.2	pgf90	same as tpr 3414
3423	5.2	pgf90	compiler abort with -mcmodel=medium -Mlarge_arrays
3426	5.2	pgCC	should support -g with --one_instantiation_per_object
3427	5.2	pgCC	handles IFSTREAM of string data improperly
3428	5.2	pgCC	same as 3427
3429	5.2	pgCC	poor C++ performance on pointer-based loops
3432			Closed Invalid
3436	5.2	all	enhancement request for -pgf77libs, -pgf90libs
3437	5.2	pgf90	signal 11 failure with compiled code
3438			Closed Invalid
3439	5.2	pgf90	pgf90 program works in 5.1, fails in 5.2

TPR	Rel	Lang/ Tool	Description
3445	6.0	pg90	ICE reshape/eoshift.
3447			Closed Invalid
3451	5.2	pgf90	documentation of TMPDIR missing
3452	5.2	pgf90	'Argument number 2 to brdf_kernel_setups: type mismatch'
3457	5.2		Closed Invalid
3461	5.2	pgCC	`pgCC trouble with STLs`
3462	5.2		Closed Invalid
3463	5.2	pgf90	program causes compiler crash
3465	5.2	pgf77	write(*,'(G20.6)') 0.0d0 has one too many zeros
3466	5.2		Closed Invalid
3468	5.2	pgcc	enum error with pgcc legal limit INT_MAX
3470	5.2	pgf90	doesn't agree with pgf77 on zero-length array handling
3471	5.2	all	-g -Mstabs caues linux compilers to fail
3475	5.2	pgf90	pgf90 program fails with 64-bit. -Mvect=sse
3479	5.2	pgf90	NULL intrinsic not PURE in pgf90
3480	6.0	pgf90	reshape error in internal procedures
3482	5.2	pgf90	pgf90 compiler TERMINATED by signal 11
3483	5.2	pgCC	pgCC gives wrong answers on 64-bit
3487	5.2	pgcc	pgcc fails to test unsigned char properly
3488	5.2	pgcc	pgcc - 1 __attribute() okay; 2 __attribute()s fails
3492	6.0	pgf90	compiler segfault with -i8 -g
3496	5.2	pgf90	Errors in Lowering ICE pgf90
3497	6.0	pgf90	use of size intrinsic as an expression for array arguments
3501	6.0	pgf90	compiler segfault -r8 -i4
3504	6.0	pgf90	use of NULL() as a component initializer
3505	6.0	pgf90	private versus public accessibility in a host and a contained procedure
3516	6.0	pgprof	pgprof causes NullPointerException
3517	6.0	pgf90	lbound/ubound when expressed as component references
3519	6.0	pgf90	pgf90 man page error
3524	6.0	pgf90	compile time - large mix of use and priv/pub statements
3539	6.0	pgf90	functions returning struct of two doubles or two longs
3548	6.0	pgcc	struct assignment - lhs is composed of subscripting, pointer indirection, and member accesses.
3549	6.0	pgcc	-g for CRAY pointers not working properly
3550	6.0	pgCC	missing -g information for C++ struct and class members
3551	6.0	pgf90	POINTER arguments passed to FUNCTIONs

TPR	Rel	Lang/ Tool	Description
3554	6.0	pgf90	same as 3551
3555	6.0	pgcc/CC	mod by a large unsigned long constant
3557	6.0	pgf90	-g for f90 pointers not working properly
3558	6.0	pgCC	64-bit pgCC link fails rel. R_X86_64_PC32 on fedora c2
3565	6.0	pgf90	USE error – Var. name same as named const
3568	6.0	pgcc	-O error w/ posfix expressions with conditional
3569	6.0	pgf90	section of assumed-size array as ptr assignment
3581	6.0	pgf90	allocatable, automatic array mix fails
3649	6.0	pgf90	program segv's when allocated array is assigned
3664	6.0	pgCC	integer divides propagate into OpenMP loops
3665	6.0	pgf90	requests to queue a duplicate destructor fail
3666	6.0	pgf90	OpenMP schedule (dynamic) no-trip loops are entered
3667	6.0	pgf90	some expressions hoisted out of a non-DO loop

4 Contact Information & Documentation

You can contact The Portland Group at:

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

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

All technical support is by e-mail or submissions using an online form at *http://www.pgroup.com/support*. Phone support is not currently available. Many questions and problems can be resolved at our frequently asked questions (FAQ) site at *http://www.pgroup.com/support/faq.htm*. Online documentation is available by pointing your browser at either your local copy of the documentation:

```
file:/usr/pgi/doc/index.htm
```

or online at *http://www.pgroup.com/doc*.