The Cray XT Compilers

Geir Johansen, *Cray Inc.*

ABSTRACT: *The Cray XT3 and Cray XT4 supports compilers from the Portland Group, PathScale, and the GNU Compiler Collection. The goal of the paper is to provide Cray XT users with an overview of the different compilers available in the Cray XT Programming Environment. Discussion will highlight the feature and performance differences between the compilers and provide guidelines in determining which compiler to use.*

KEYWORDS: Cray XT3, Cray XT4, Compilers

1.0 Introduction

The goal of this paper is give an overview of the compilers available for the Cray XT3 and Cray XT4 systems. The three different compilers available for building code on the Cray XT are: The Portland Group (PGI) compilers, the GNU compiler collection (GCC), and the PathScale Compile Suite. The compiler environments are similar in that they all provide:

- C/C++ compiler
- Fortran 90/95 compiler
- AMD64 code generation with SSE2 support
- OpenMP support

Cray Inc. is fortunate in that all three of the compilers are being actively developed and supported. All three compilers have had significant feature enhancements and performance improvements over the life of the Cray XT product. As a result of this dynamic environment, an important caveat to this paper is that it represents a current snapshot and the information provided is subject to change.

2.0 Cray XT Programming Environment

The programming environment for Cray system is essentially a cross compiler environment. The compiler and linker are executed on Cray XT login service nodes, while the resulting executables are invoked on Cray XT compute nodes. The operating system running on the compute nodes is either the Catamount microkernel or Cray Compute-Node-Linux (CNL), so the executables need to be linked with the appropriate system libraries for execution on the compute nodes. The compilers also need to include the communication and scientific libraries that are built for each specific compiler. For example, there are separate MPICH libraries built for each of the three compilers supported on the Cray XT system. The Cray XT Programming Environment uses the *module* utility and compiler scripts to ensure the executable is built with the correct compiler options, header files, and libraries appropriate for the compiler.

2.1 Modules

Similar to other Cray Inc. systems, the Cray XT uses the *modules* utility to initialize the programming environment for the user. The *modules* utility will set the appropriate environment variables so the compilers will find the correct header files and libraries to create an executable for the Cray XT compute nodes. The modules *PrgEnv-pgi*, *PrgEnv-gnu*, and *PrgEnv-pathscale* are the main module files for each of the three compilers. When the main module file is loaded, it will proceed to load the other programming environment modules that are needed to build code for the Cray XT. The following table is a list of the Cray XT programming environment module files:

Module and Package Name	Description
PrgEnv-pgi *PrgEnv-gcc* *PrgEnv-pathscale*	Main programming environment module that loads other programming environment modules.
pgi	PGI compilers
gcc	GCC compilers
pathscale	PathScale compilers
xt-pe	Compiler driver scripts
xt-mpt	MPICH2 Message Passing Interface 2 (MPI-2) library and SHMEM library
acml	AMD Core Math Library
xt-libsci	Cray XT3 LibSci scientific library routines
iobuf	Cray I/O buffering library for Catamount [Not part of the *PrgEnv* module.]
fftw	Fast Fourier Transform routines [Not part of the *PrgEnv* module.]
craypat	Cray Performance Analysis Tool [Not part of the *PrgEnv* module.]
totalview	TotalView debugger [Not part of the *PrgEnv* module.]

Table 1. Programming Environment *modules* files

2.2 Compiler Drivers

The compiler commands (see Table 2) are shell scripts that read in the environment variables that have been initialized by modules files and proceed to call the compiler executable with the appropriate arguments. Only the listed compiler commands should be used to compile code targeted for the compute nodes. Using another compiler shell script (i.e. *mpicc*) or calling the compiler directly could result in an important option being missed that is essential for execution on the compute nodes. For example, calling *pgf90* to compile and link a code with the PGI Fortran compiler will result in an executable that does not have the appropriate libraries to run and communicate on the Cray XT compute nodes.

Compiler Command	Compiler
cc	C compiler
CC	C++ compiler
ftn	Fortran 90/95 compiler
f77	Fortran 77 compiler (only valid for PGI and GCC 3.x)

Table 2. Compiler commands

2.3 Example use of module and compiler

The module environment allows the user to switch from one compiler to another. The following example shows a user changing a compiler, and then changing the release version of the compiler:

```
$ module load PrgEnv-pathscale
$ ftn -version
/opt/xt-pe/2.0.03/bin/snos64/ftn: INFO: catamount target is being used
PathScale EKOPath(TM) Compiler Suite: Version 2.5
Built on: 2006-08-22 21:02:46 -0700
Thread model: posix
GNU gcc version 3.3.1 (PathScale 2.5 driver)
Copyright 2000, 2001 Silicon Graphics, Inc. All Rights Reserved.
Copyright 2002, 2003, 2004, 2005, 2006 PathScale, Inc. All Rights Reserved
See complete copyright, patent and legal notices in the
/opt/pathscale/share/doc/pathscale-compilers-2.5/LEGAL.pdf file.
$ module swap PrgEnv-pathscale PrgEnv-pgi
$ ftn -V
/opt/xt-pe/2.0.03/bin/snos64/ftn: INFO: catamount target is being used
pgf90 7.0-2 64-bit target on x86-64 Linux
Copyright 1989-2000, The Portland Group, Inc. All Rights Reserved.
Copyright 2000-2007, STMicroelectronics, Inc. All Rights Reserved.
$ module swap pgi/7.0.2 pgi/7.0.3
$ ftn -V
/opt/xt-pe/2.0.03/bin/snos64/ftn: INFO: catamount target is being used
pgf90 7.0-3 64-bit target on x86-64 Linux
Copyright 1989-2000, The Portland Group, Inc. All Rights Reserved.
Copyright 2000-2007, STMicroelectronics, Inc. All Rights Reserved.
$
```

2.4 Using Multiple Compilers for an Application

Each of the programming environment libraries (mpich, shmem, libsci, and acml) has a version that is built using each of the compilers. A reason for this is because of incompatibilities with Fortran modules between the different compilers. As a result, it is not recommended to use different Fortran compilers to build the object files of an executable. Likewise, incompatibilities in C++ name mangling and the C++ library makes it not possible to use different C++ compilers in building the same executable. There are no issues with using different C compilers in building an application.

2.5 Special Catamount Compiler Libraries

The Catamount microkernel supports a subset of the library routines that are supported by Linux. As a result, there are cases where compiler libraries provided by the compiler vendor has calls to routines that are not supported by the Catamount microkernel. For example, the PathScale Fortran library *libpathfstart.a* calls the routine *sigaltstack*, which is not supported by the Catamount microkernel. PGI and PathScale have provided Cray with modified libraries that only call routines that are supported by the Catamount microkernel. The following is the list of compiler libraries that were modified for the Catamount microkernel:

PGI

- *libpgc.a*
- *libpgf90.a*
- *libpgftnrtl.a*
- *libC.a*

PathScale

- *libpathfstart.a*
- *libpathfortran.a*
- *libpathfortran_p.a*

The compiler scripts will locate the Catamount programming versions of these libraries. One area where this may be a problem is when the libraries are explicitly stated on the link command line. For example, a code with both C and Fortran code that is being linked by the PGI C compiler may contain the options '*-lpgf90 -lpgftnrtl*' on the command line to link in the PGI Fortran libraries. In the current Cray XT 1.5 release, the user would need to specify '*-lqk_pgf90 –lqk_pgftnrtl'* to link the program with Catamount (qk) versions of the libraries. Future versions of Cray XT will resolve this issue, so the user will not have to change their build scripts to specify the 'qk' library names.

2.6 Compiling Code for Service Nodes

In order to compile code, such as a utility program, that is to be executed on a login/service node the compilers should be called directly. For example, to compiler a C code, the PGI C compiler *pgcc*, the GNU C compiler *gcc*, or the PathScale C compiler *pathcc* can be invoked to compile the code. These compilers will find the appropriate header files and libraries in their normal Linux locations.

3. The Portland Group Compilers

3.1 Cray XT3 Usage

The PGI compilers were the only supported compilers for the Cray XT3 when the system was originally released. The PGI compilers were chosen because they provided a good combination of features and performance for HPC programming. The initial release for the Cray XT3, version 5.2, lacked C99 and C++98 standard features. These issues have all been resolved and PGI produces good performing C and C++ code. The current version of PGI that is supported on the Cray XT is version 7.0-3.

The PGI compiler has been by far the most used compiler for generating code for the Cray XT. This extra exposure time has been helpful in resolving issues related to libraries and tools used with the compiler. One notable example of this is the *ftn* compiler script '*-default64*' option, which is used to load libraries that were compiled with the '*-i8 –r8*' options. Currently, this option is only supported for the PGI compiler. As a result of PGI being the most used compiler, Cray Inc. software development group has focused efforts on getting programming environment software, such as IOBUF and CrayPat, to work with PGI compiled code. Cray Inc. software development group is working on getting the libraries and tools to perform equally as well with the other compilers.

3.2 Compiler Options

The PGI compiler has many options to specify features and optimization techniques to be performed by the compiler. Chapter 2 of the *PGI User's Guide* provides a good overview of optimization options available for the PGI compilers. The PGI web site (www.pgoup.com) has a page that discusses the porting and tuning of specific HPC applications, such as GAMESS and WRF. Another good resource for suggested PGI options is the SPEC CPU2006 Benchmark web page (www.spec.org/cpu2006), which shows the options used for each of the benchmark results that were submitted using the PGI compiler. The following are frequently used PGI options

- ***-fast*** This flag is a collection of optimization options. In PGI 6.2 and earlier releases, the user should use the option *–fastsse*. The specific optimization flags that are specified by the *–fast* are: *-O2 -Munroll=c:1 -Mnoframe –Mlre -Mvect=sse –Mscalarsse -Mcache_align –Mflushz*
- ***-fastsse*** In PGI 7.0 this option is identical to ***-fast***
- ***-Mipa=fast,inline*** Invokes inter-procedural analysis. The fast option is collection of IPA sub-options that are generally optimal for the targeted machine.
- ***-O3, -O4*** The *-fast* option contains *'-O2'*, so this option must appear after the *–fast* option on the command line
- ***-Minfo*** Outputs messages of the optimizations performed by the compiler.

The following PGI options have also been observed in compiling code for the Cray XT3:

- ***-Mnontemporal*** Informs compiler to force generation of nontemporal move and prefetch instructions.
- ***-Mprefetch=distance:8,nta*** *distance* option for the *prefetch* flag sets the fetch-ahead distance to 8 cache lines. The *nta* option instructs compiler to use the *prefetchnta* instruction.
- ***-Msafeptr=all*** Optimization option that instructs the compiler that pointers do not have data dependencies.
- ***-Munroll=n:X*** Instruct the compiler on the number of times a loop should be unrolled
- ***-Minline=levels:X*** Informs the inliner to perform X levels of inlining, where the default is 1. This is an important option for C++ code. The PGI User Guide suggests using *–Minline-levels:10* for C++ code.
- ***-Kieee*** Floating-point operations are performed in conformance with the IEEE 754 standard. This option is useful for producing bit identical results. The use of this option will likely result in less performance.
- ***-Mneginfo*** Outputs messages on why certain optimizations were not performed.
- ***-Mnodepchk*** The compiler assumes that potential data dependencies do not conflict. Option can produce incorrect code if there are data dependencies.
  -help Displays useful information about the options specified on the command line**.**

4. GNU Compiler Collection

4.1 Cray XT3 Usage

The GNU compiler collection is the open source compiler made available by the GNU project. The GCC compiler is used to build the system software for the Cray XT systems. The GCC compiler became available for building compute node executables in Cray XT OS 1.2. This version, GCC 3.2.3, did not support the compilation of Fortran 90 code. The GCC compiler was used to compile C and C++ code, as it was shown to have some feature and performance advantages over earlier PGI releases

The 4.1 release of GCC has Fortran 90/95 capability. This release also contains a subset of the new Fortran 2003 features. Users have found the GCC 4.1 C compiler enforces language syntax more strictly than the 3.x versions. The current release of the GCC compiler is 4.1.2.

4.2 Compiler Options

The following GCC compiler options are essential in producing optimize code with the GCC compiler:

- ***-O3*** Turns on most of the useful optimization features for GCC.
- ***-ffast-math*** An important optimization option that is vital for optimization. It is not included in *–O3* because it could result in math functions not complying with IEEE or ISO standards.

The following GCC options have been used on the Cray XT3 to improve performance:

- ***-fprefetch-loop-arrays*** Generate instructions to prefetch memory to improve loop performance.
- ***-funroll-loops*** Enable loop unrolling optimization
- ***-ftree-vectorize*** Enable tree vectorization

5. PathScale Compiler Suite

5.1 Cray XT3 Usage

Qlogic PathScale Compiler Suite targets the high performance computing market. The compiler has Cray roots in that its Fortran front-end is based on the Cray front-end that was made available by SGI. A benefit of using the Cray front-end is that the PathScale Fortran compiler supports the *assign* command, which is used to customize Fortran I/O. The *assign* command has been shown to improve the performance of Fortran file I/O running on Catamount microkernel compute nodes. For C and C++ code, PathScale uses the GNU front-end.

The PathScale compiler was not originally supported in the Cray XT software. A main reason this was done was to reduce complexity in early releases of the Cray XT3 software, since a set of communication (MPICH, SHMEM) and scientific (libsci, acml) libraries is needed for each compiler, The PathScale compiler is now supported on Cray XT systems and can be ordered and licensed through PathScale. The current version supported on the Cray XT is PathScale 3.0.

5.2 Compiler Options

The PathScale *pathopt2* tool can be used to find the optimal set of options for compiling a program. The tool accomplishes this by iteratively testing different option combinations using an adaptive process. This tool is described in chapter 7 of the PathScale Compiler Suite User Guide. This chapter provides detail information on PathScale optimization features, while chapter 6 gives a quick overview of optimization options. The *eko* (Every Known Optimization) man page contains a description of the optimization options for the PathScale compilers. A source for suggestions on PathScale compiler optimization options is the SPEC CPU2006 Benchmark web page (www.spec.org/cpu2006), which provides information on the options used for each of the benchmark results that used the PathScale compiler.

The PathScale option ***–Ofast*** is a collection of the optimization options. A user may want to first use a subset of the ***–Ofast*** options to ensure correctness of results. The option ***–Ofast*** specifies:

- ***–O3*** Aggressive optimization
- ***-ipa*** Turns on inter-procedural analysis (IPA).
- **–OPT:Ofast** Specifies the optimization options *-OPT:ro=2:Olimit=0:div_split=ON:alias=typed*
- ***–fno-math-errno*** ERRNO is not set after calling math functions
- ***–ffast-math*** Improves performance of floating point math. Math results may not conform to the IEEE standard.

The following options have been used on Cray XT3 systems to improve the performance of PathScale executables:

- ***-CG:use_prefetchnta=ON*** Instruct the compiler to use the *prefetchnta* instruction.
- ***-CG:movnti*** Perform non-temporal stores
- ***-O2*** Default optimization level
- ***-LNO:fu=X:full_unroll_size=Y*** Parameters for loop unrolling. Default values are trip count X is 5 and unroll loop size Y is 2000.
- ***-LNO:simd=2*** Aggressive vectorization option
- ***-LNO:vintr=2*** Aggressive loop vectorization
- ***-OPT:alias=restrict*** Pointers are assumed to be non-overlapping.
- ***-OPT:ro=3*** Allow more optimizations that may affect floating point results
- ***-OPT:unroll_size=256*** Set the maximum number instructions of an unrolled inner loop (default is 40)
- ***-OPT:recip=ON:fast_sqrt=ON*** Use reciprocals in math calculations

6. Performance

Performance has improved with each new release of the compilers. It is important when viewing published performance results to note the version of the compiler that was being used.

6.1 HPCC Challenge Benchmark

The HPCC Challenge (HPCC) benchmark was used to test the C compilers. The results (see Appendix A) show that for most of the tests the results were very similar. One exception is the STREAM benchmark, where the GCC compiled did not perform as well as the others. The GCC compiler does not have an option to specify the use of the *prefetchnta* instruction.

The HPCC benchmark showed an interesting case for the PathScale compiler. The RandomAccess kernel performed much better when it was compiled using the '*-O2*' option rather than the '*-O3*' option. Examining the code that was generated showed the '*-O3*' version had loop scheduling code that did not add performance to the loop.

The results from the HPCC benchmark indicate that each of the C compilers produce similarly performing code.

6.2 Polyhedron 2005 Fortran Benchmark

The Polyhedron 2005 Fortran Benchmark (www.polyhedron.com) was used to test the Fortran compilers. The results (see Appendix B) clearly show the GCC Fortran compiler not performing as well as the PGI and PathScale Fortran compilers. The geometric mean for the GCC results showed 25% less performance than the PathScale compiler. Two exceptions where GCC Fortran clearly outperformed PGI and PathScale were the Channel and Linpack benchmarks.

The benchmark results showed the PathScale Fortran compiler slightly outperforming the PGI compiler. The difference in the geometric mean was less than 5%.

6.3 Stepanov Benchmark

The Stepanov benchmark is used to measure the level of abstraction that C++ constructs add to a code as compared to a C code performing the same functionality. The benchmark showed that the GCC C++ having the best results with PathScale C++ compiler being a close second. The PGI C++ compiler did not perform very well with this benchmark.

6.4 Cray Inc. Application and Benchmark Groups

Members of the Cray Inc. Application and Benchmark groups were interviewed to gain their perspective and experience of the three compilers. The following is a consensus of what they reported:

1. **PGI and PathScale are both used for Fortran code:** The GCC Fortran compiler is never used to compile Fortran. The groups' experience is that the PGI compiler will perform better for some Fortran codes, while the PathScale compiler performs better for others. The performance difference is almost always with 10% of each other. They reported little difficulty with switching from one compiler to another.
2. **GCC is generally used for C and C++ code:** The experience is that code compiled with GCC C compiler typically performs as well as the other compilers, and sometimes much better. The GCC C++ compiler is almost always used to compiler C++ code.
3. **Better compiler directives are needed for PGI and PathScale:** The programmers have had mixed results in using the PGI and PathScale compiler directives to optimize specific sections of code. They felt that both PGI and PathScale could greatly improve the effectiveness of their compiler directives.

7. Guidelines in Choosing a Compiler

7.1 Fortran Performance

The PGI and PathScale compilers both produce consistently better performing Fortran code than the GCC Fortran compiler. For most case the PGI and/or PathScale compilers should be used for Fortran programs.

7.2 C and C++ Performance

All three compilers have been used to compile C code with good results. The GCC compiler is used to build operating systems and system utilities, and generates highly optimized code for this type of software. A Fortran program that has some C routines to interface with the operating system, may want to use the PGI or PathScale compiler for the Fortran code and the GCC compiler for the C code.

The experience of the Cray Inc. Applications and Benchmark groups is that GCC C++ produces the best performing C++ code. The Stepanov benchmark supports this assertion.

7.3 Cray XT Exposure Time for PGI

The PGI compiler was the first officially supported compiler for the Cray XT3 and has had much more exposure in compiling codes for the Cray XT. A reason this is an advantage is because of the greater experience of using the PGI code with the Cray XT3 libraries and tools. The scientific and communication libraries are all compiled using each of the compilers, so the PGI compiled versions of these libraries have had more exposure. Currently, PGI is the only compiler with Cray programming libraries that support the *ftn* '*-default64*' option.

Programming environment tools, such a CrayPat and TotalView, is another area where more collective experience in using PGI compiled code is an advantage for using the PGI compiler. Compiler related issues that cause problems with tools are more likely to have been addressed for PGI compiled code. For example, at the current moment there is a significant problem in using CrayPat to analyse C++ code compiled with GCC.

7.4 ISV Application Recommendations

All three compilers are widely used on other HPC machines using the AMD Opteron, so when porting a HPC code to the Cray XT machine it is very likely to have been built on another platform that uses the AMD Opteron. The ISV application developers may have recommendations on which compiler and compiler options to use to get the best performance from the

application. The PGI website has a page with porting and tuning suggestions for common HPC applications.

7.5 IOBUF vs. Assign

The Cray I/O Buffering library can greatly improve I/O performance of codes running on Catamount microkernel compute nodes. PathScale uses system calls rather than libraries routine to implement Fortran I/O to files, so IOBUF is less effective in improving the Fortran file I/O performance of PathScale compiled code. PathScale Fortran I/O to standard input/output does not have this issue, so IOBUF can be used to improve this type of I/O.

The PathScale Fortran front-end is based on the Cray Fortran front-end, so it supports the *assign* command. The *assign* command allows the customization of Fortran I/O. The use of the *assign* '*-b*' option has been shown to improved Fortran file I/O performance of PathScale compiled codes.

7.6 Performance Support

The two commercial compiler vendors PGI and PathScale compete mainly on performance. They have a vested interest in improving the performance of their compiled code, so a good part of their development focus is on performance issues. If there is a performance issue with a HPC program running on a Cray XT system, both PGI and PathScale are more likely to be responsive than the GCC open source community in resolving the issue

7.7 Portability of GCC C and C++

The GNU C and C++ compiler are de facto standards for C and C++ code. Some C and C++ codes may expect GCC features and behavior, such as *gnu attributes* directives that are not supported by the other C and C++ compilers. In this situation, the GCC compilers may need to be used.

7.8 Fortran 2003 Standard

The GCC Fortran compiler *gfortran* has implemented more of the new Fortran 2003 features than either PGI or PathScale. Codes that take advantage of new Fortran 2003 features may need to use the GCC Fortran compiler. PGI and PathScale are both working on implementing the new Fortran 2003 features.

Conclusion

Cray XT provides a programming environment that supports multiple compilers and provides a method to easily switch between the compilers. The PGI or PathScale compiler should be used for Fortran, while all three compilers can be used for C code. Unless there is an explicit reason to choose a specific compiler, a user may want to start with the PGI compiler to take advantage of its Cray XT experience. Once the code is successfully executing on the Cray XT using the PGI compiler, the other compilers can be used to check for any difference in performance.

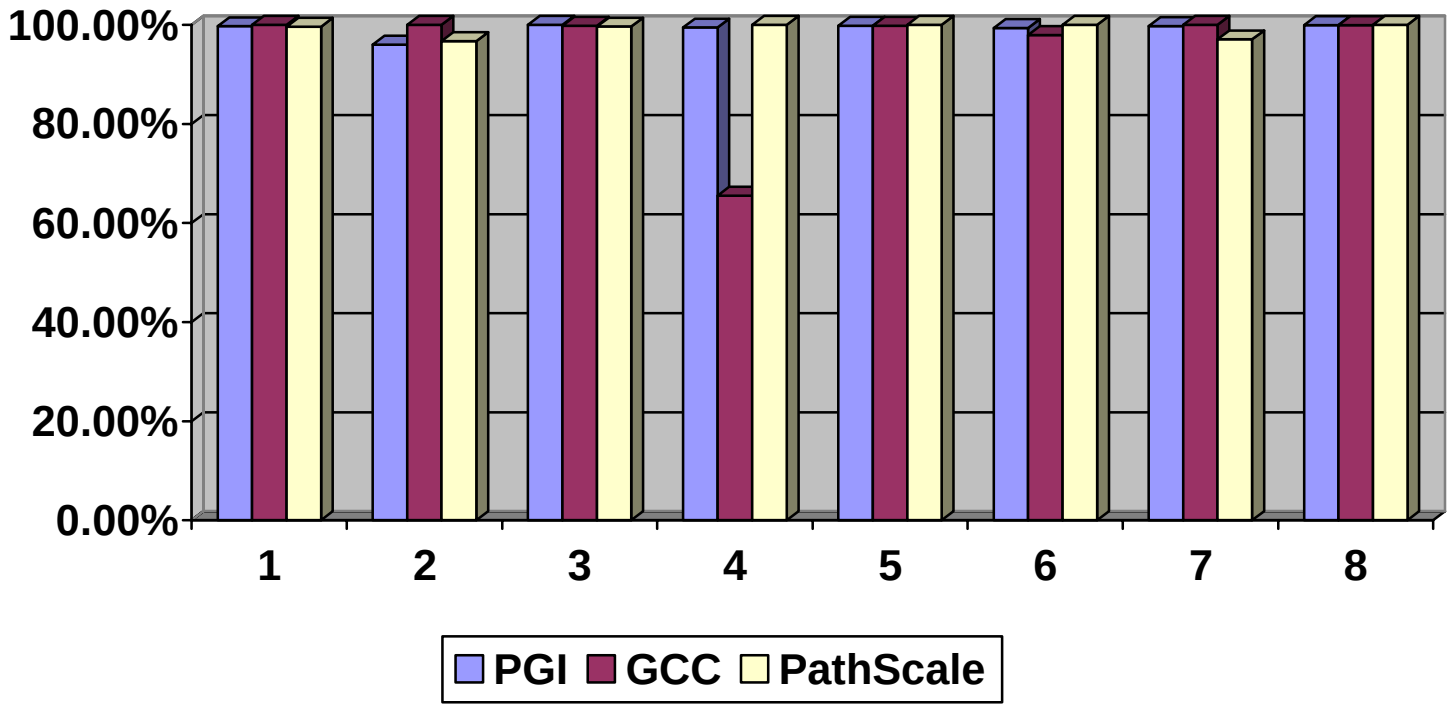
Since the initial product release of the Cray XT3, there have been significant improvements to each of the compilers available for the Cray XT systems. While all the compilers work well, there are opportunities for improvement for each of the compilers in there ability to build HPC code. The current competitive environment among the compilers bodes well for future performance and feature enhancements to the compilers supported on Cray XT systems

About the Author

Geir Johansen works in Software Product Support, Cray Inc. He is responsible for support of C, C++, libc, MPI, SHMEM, TotalView and other debuggers, and performance tools for the Cray X1 and Cray XT platforms. He can be reached at Cray Inc., 1340 Mendota Heights Road, Mendota Heights, MN 55120, USA; Email: geir@cray.com

Appendix A

HPC Challenge Benchmark Results

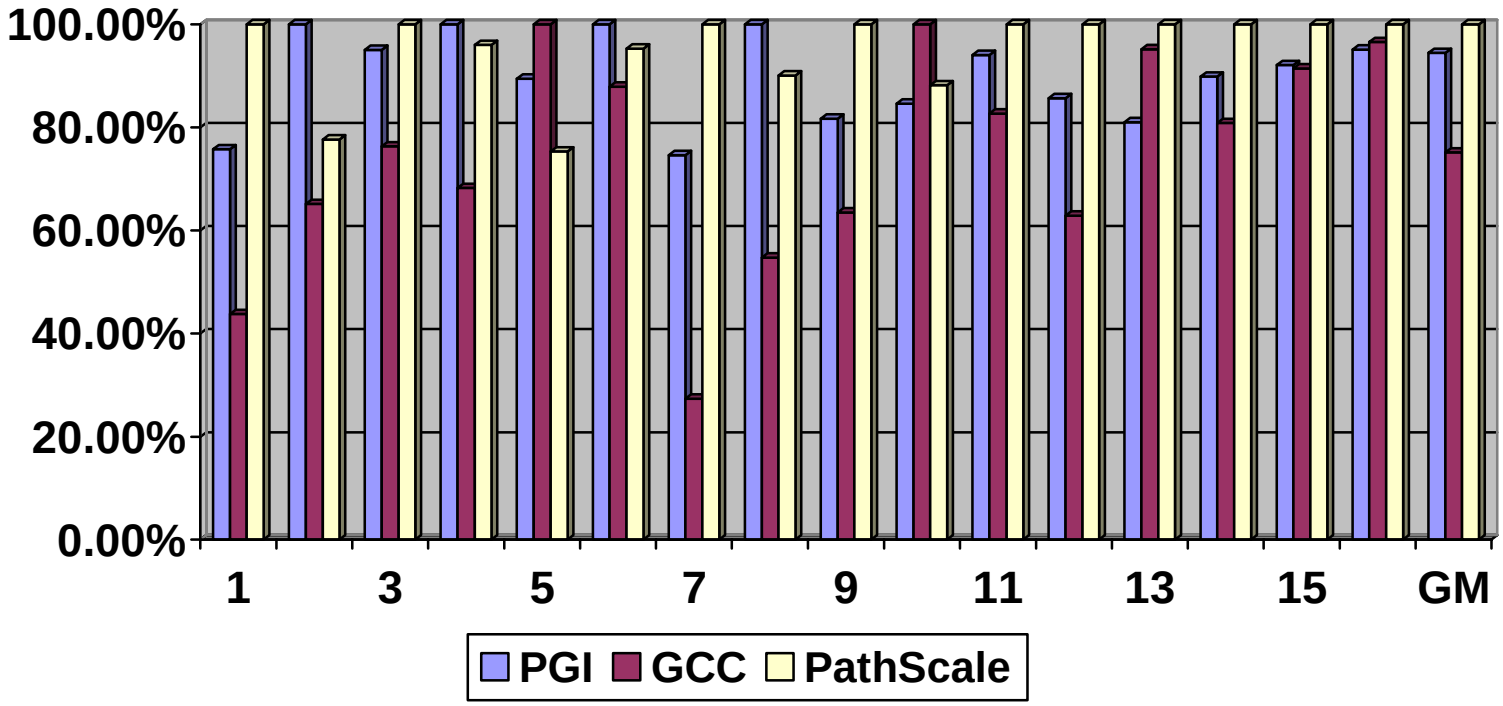


The percentage value is the percentage that compiler performed in comparison to the best performing compiler for that test.

1) PTRANS
2) HPL
3) DGEMM
4) STREAM Triad
5) Random Access
6) FFTE
7) RandomRing Latency
8) RandomRing Bandwidth

Appendix B

Polyhedron 2005 Fortran Benchmark Results



The percentage value is the percentage that compiler performed in comparison to the best performing compiler for that test.

1) ac
2) aermod
3) air
4) capacita
5) channel
6) doduc
7) fatigue
8) gas_dyn
9) induct
10) linpk
11) mdbx
12) nf
13) protein
14) rnflow
15) test_fpu
16) tfft

GM is the Geometric Mean
Benchmark originated from www.polyhedron.com
Benchmark was run on a Cray XT3 system with 2.4 MHz processor. Date of runs: 4/28/07.
GCC version 4.1.2; PathScale version 3.0; PGI version 7.0-3