Evaluation of C++ Compilers for the Cray T3E

Hans-Hermann Frese *Detlef Reichardt*
Philipp Rohwetter

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
Berlin, Germany

Abstract

The C++ compiler provided with the Cray Programming Environment 3.5 is investigated with respect to conformance with the ISO/IEC 14882:1998 standard and availability of the new C++ standard library including the Standard Template Library (STL). Comparison is made with other C++ compilers for the Cray T3E.

1 Introduction

Object oriented programming with C++ [2, 4, 21] is gaining increasing interest for high-performance computing applications on massively parallel computers [36, 38]. This arises mainly due to the possibility of combining the strength of an object oriented language with respect to efficient program development with the computing power of massively parallel systems. For porting C++ programs onto systems like Cray T3E the standard conformance of C++ compilers as well as the availability and efficient implementation of (standard) libraries play an important role. These issues will be addressed in the present paper.

After a short review of C++ history the main concepts of the language are summarized. The Cray C++ Compiler Release 3.5 is evaluated with respect to standard conformance and compared with the well known KAI C++ compiler. It will be shown that the Cray C++ compiler is mostly standard conforming with respect to the core language whereas the accompanying C++ library does not meet the recent ISO/IEC standard [12]. This has been a serious obstacle for running C++ applications developed on other standard conforming programming environments (e.g. GNU C++) on the Cray T3E.

We will show that with our implementation of the STLport library [10, 32] for the Cray T3E the standard conformance is enhanced to meet the requirements of portable C++ applications. Therefore, the combination of the Cray C++ compiler and the STLport library for the Cray T3E provides an efficient programming environment for portable object oriented programs on a massively parallel system.

2 C++ History

In 1983 a research group at AT&T Bell Laboratories headed by Bjarne Stroustrup started to develop the new programming language C++ [33, 34, 35]. The goal was to facilitate the development of C programs through an enhanced support of the programmer, especially by stronger type checking and by making object oriented methods available [13, 26, 27]. Additionally, the efficiency of C [11, 15, 16] should not deteriorate and the compatibility to C should be maintained. However, the necessary compromise between efficiency, compatibility with C, and object oriented paradigms lead to some restrictions for the feasible flexibility and the implementation of object oriented paradigms.

C++ presents an enhanced data type concept over C. It also incorporates the class concept from Simula 67 [8] and operator overloading from Algol 68 [37]. Early versions of the new programming language were called “C with classes”. Later, the name was changed to “C++” due to a proposal by Rick Mascitte with reference to the increment operator “++” in C.

The ANSI committee X3.J16 established in 1989 completed the ISO/IEC Standard 14882:1998 for C++ [12] in 1998. Because this process took nine years the *Annotated C++ Reference Manual* (ARM) [9] published in 1990 became a de-facto standard which served as a basis for compiler and library development in the meantime. However, several features added later on to the core language, namely the concepts of templates and namespaces, lead to a complete redesign of the C++ standard library containing the so called *Standard Template Library* (STL) [1, 24] as its central part. Hence, in most cases a reimplementation of the standard library is necessary to meet the recent ISO/IEC standard. Therefore, we have the temporary situation of dealing with "old" and "new" versions of the C++ standard library, which are incompatible with each other.

3 Properties of Object Oriented Programming and C++

The design goals of C++ were the easier development of C programs and an improved support for programmers. The new fundamental concepts in C++ are:

- *Objects* which communicate through messages and which represent a model of the real world through their relations.
- The *class concept* defines user specific data types, related operators, and access functions. The *data encapsulation* guarantees that private objects can exclusively be accessed through specified methods.
- *Inheritance* allows the specification of relations between classes in the sense of abstraction or specialization. Inheritance also helps to avoid multiple implementations.
- A *strong type concept* inhibits undesired side effects for implicit type conversions like in C. The programmer has to specify a type conversion explicitly in C++.
- *Function and operator overloading* allow to define existing functions for additional parameter types. The strong type concept allows a function call only with the parameter types defined for the function.
- *Efficiency*: except for virtual functions, the conceptual extensions in C++ do not cause any runtime delays compared to C. Even for heavily used virtual functions in class libraries there is only a small loss in efficiency.

C++ is a "hybrid" programming language. It includes the programming language C as a subset. Because C++ is fully compatible with C, conventional C programs or parts thereof can easily be incorporated in C++ and enhanced through the new concepts.

The newly developed components can be implemented completely in an object oriented manner. The advantages of object oriented programming in C++ are:

- The *reuse* of modules and program skeletons increases the development productivity and the program's quality due to the reuse of tested components.
- Clearly arranged *interfaces* facilitate the expandability and the maintenance of software products.
- The *hierarchical* structure of classes, variables, and methods reduces the complexibility both of single components and of the entire system.
- The solution, i.e. the desired software product, can easily be derived from the problem.
- In general, object oriented programming languages provide a higher *abstraction level* through the concepts of inheritance, polymorphism, and dynamic binding than traditional programming languages.

In addition to the earlier concepts of C++, the ISO/IEC standard added the features:

- *templates* which were derived from Ada generics
- multiple *inheritance*
- static member functions
- pure virtual functions
- *overloading* of the operators '`->`', '`new`', and '`delete`'
- *name spaces* provide type safe binding of C++ libraries to avoid name collisions.

4 Cray C/C++ Programming Environment

The Cray C/C++ Programming Environment [7] provides an integrated environment for the development of C++ applications. The main components of the Cray C/C++ Programming Environment include:

- the native Cray C++ compiler
- the Cray Standard C compiler
- the CrayLibs package including the C++ standard library
- the CrayTools package including debugging and performance analysis tools.

4.1 Cray C++ Compiler

As stated in the reference manual [5] the Cray C++ Compiler Release 3.5 accepts almost the C++ language as defined by the ISO/IEC standard. The compiler itself consists of a preprocessor, a language parser, a prelinker, an optimizer, and a code generator.

Whereas early versions of the Cray C++ compiler generated intermediate C code which was then compiled to machine code by the Cray Standard C compiler, later releases now compile C++ programs to machine code directly. For historical reasons, the Cray C++ compiler also has a cfront compatibility mode which also to continue to use old C++ code developed with the early versions of the Cray C++ compiler.

4.2 Standard Conformance

Our evaluation results in section 8 show that the latest release 3.5 of the Cray C++ compiler conforms very well with the C++ core language defined in the ISO/IEC standard with only a few deviations. The unsupported features from the C++ core language are documented in the Cray C++ Compiler Reference Manual [5]:

- `reinterpret_cast` does not allow casting a pointer to a member of one class to a pointer to a member of another class if the classes are unrelated.
- Two-phase name binding in templates are not implemented.
- Putting a `try/catch` around the initializers and body of a constructor is not implemented.
- Template `template` parameters are not implemented.
- Universal character set escapes are not implemented.
- The `export` keyword for templates is not implemented.
- `extern inline` functions are not supported.
- Covariant return types on overriding virtual functions are not supported.

Whereas for the C++ standard library, there are many deviations from the ISO/IEC C++ standard (see section 8).

5 KAI C++ Compiler (KCC)

In 1979 Kuck & Associates, Inc. (KAI) was founded by David Kuck as a a privately held company. KAI Software is now a divison of Intel Americas, Inc.

Amongst various other well known products like the KAP/Pro Toolset and Visual Kap, KAI Software have developed a very well known C++ compiler [18] which is highly accepted by the C++ community due to its conformance to the ISO/IEC standard. The KAI C++ compiler (KCC) has been ported to various platforms including the Cray T3E [17].

The following features of the ISO/IEC C++ standard are not supported by the KCC compiler [19]:

- Two-phase name binding in templates are not supported.
- A partial specialization of a class member template cannot be added outside of the class definition.
- Universal character set escapes are not supported.
- The `export` keyword for templates is not supported.

In addition to the standard conformance for the C++ core language, KCC comes with a mostly ISO conforming C++ class library and POSIX thread support.

Our evaluation results in section 8 show that the KCC compiler is only slightly superior over the Cray C++ compiler with respect to the core C++ language.

6 The C++ Standard Library

In addition to the development and standardization of C++, a standard library was developed to supplement the C++ compiler. The C++ standard library [14] extends the capabilities of C++ without blowing out the compiler. Therefore, the C++ standard library provides a new level of abstraction.

The C++ standard library consists of specialised libraries to provide standardized methods for three main areas:

- the *Standard Template Library* (STL)
- the `string` classes for string manipulation
- the `iostream` library for input/output of data.

In addition, the C++ standard library uses special mechanisms and templates heavily.

6.1 The Standard Template Library (STL)

The Standard Template Library (STL) [1, 3, 23, 28, 30] forms the heart of the C++ standard library. The STL provides solutions for arbitrary definable element types and forms a new abstraction level.

The STL is based on a teamwork of well structured components:

- *Containers* hold objects of a certain type. Different container classes provide different techniques for the administration of objects, e.g. arrays, linked lists, or keys.
- *Iterators* provide a mean to iterate over a quantity of objects. As a decisive advantage, iterators provide the same interface to access the elements for all containers.
- *Algorithms* process elements in quantities and quantities as a whole. Algorithms can sort, search, delete, or modify elements using iterators. Therefore, algorithms do not have to be implemented for every quantity class again.

6.2 Strings

The `string` classes allow to handle strings like fundamental data types. Memory is allocated and returned automatically. The corresponding operators can be used for assignements and comparisons.

6.3 IOStreams

The `stream` classes [20] form one of the most important parts of the C++ standard library. Every input/output stream consists of a stream of data. Streams are objects whose characteristics are defined in classes. Different global objects are defined for the standard input and output channels.

6.4 Standard Conformance

The C++ standard library which comes with the KCC compiler is based on the Modena C++ Standard Library [22]. Our evaluation results in section 8 show that this library conforms very well with the ISO/IEC C++ standard.

In contrast, the C++ standard library included with the Cray C++ Programming Environment Release 3.5 has many deviations from the ISO/IEC C++ standard.

7 STLport

In 1997 Boris Fomitchev [10] started to port the SGI STL [28] to gcc and Sun SPARC. A few months later, it was ported to other systems. As this was a portable implementation of the STL, it was called *STLport*.

To overcome the deviations from the ISO/IEC C++ standard for the Cray C++ Programming Environment, we decided to port the STLport library [31, 32] to the Cray T3E operated at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). The port itself was pretty difficult but with several code changes we finally managed to get it through the compiler. Unfortunately, we did not succeed

to implement `locales` for the Cray T3E. This was judged as a minor problem because HPC number crunching rarely requires the use of multilingual character sets in foreign languages.

Our evaluation results in the next section show that the Cray C++ compiler in combination with the STLport standard library is a perfect tool for portable standardized object oriented programming The new environment conforms almost as good to the IEEE/IEC C++ standard [12] as the KCC compiler.

This combination enables the user to portable and standard conforming programming on our Cray T3E.

8 C++ Compiler and Library Comparison

In [29] Ross Smith provides a detailed comparison chart for different C++ compilers and standard libraries. Our evaluation for the Cray T3E in table 1 on the following page shows that there are only minor differences between the Cray C++ compiler and the KAI C++ compiler. The Cray compiler does not accept covariant returns and exception throws according to the recent ISO/IEC standard. In addition, both compilers do not allow templates as parameters.

The evaluation of the standard library provided with the Cray C++ Programming Environment shows that it does not conform with the recent ISO/IEC standard mainly because it is based on the old ARM. Whereas, the standard library provided with the KCC compiler mostly comforms with the ISO/IEC standard. With our implementation of the STLport standard library the conformance of the Cray C++ programming environment can be enhanced significantly. The only deviations from the standard are iostream templates, locales, and string streams. This was judged as a minor deficiancy for a massively parallel system like the Cray T3E because the output of language specific special characters is not such an important issue for high-performance number crunching applications.

Overall our evaluation shows that the combination of the Cray C++ compiler and the STLport standard library conforms very well with the ISO/IEC C++ standard and that the combination is competitive with the KCC compiler.

9 Conclusions

We have shown that with our implementation of the STLport library for the Cray T3E the standard conformance of the C++ programming environment is enhanced to meet the requirements of portable C++ applications. Therefore, the combination of the Cray C++ compiler and the STLport library for the Cray T3E provides an efficient programming environment for portable object oriented programs on a massively parallel system.

References

[1] Matthew Austern. The sgi standard template library. *Dr. Dobb's Journal*, August 1997.

[2] Grady Booch. *Object-oriented Analysis and Design.* Benjamin/Cummings, 2nd edition, 1994.

[3] Ulrich Breymann. *Designing Components with the C++ STL: A New Approach to Programming.* Addison-Wesley, 2nd edition, 2000.

[4] Timothy Budd. *Introduction to Object-Oriented Programming.* Addison-Wesley, 2nd edition, 1997.

[5] Cray Inc. *Cray Standard C and Cray C++ Reference Manual.* 004-2179-005.

[6] Cray Inc. *CRAY T3E C and C++ Optimization Guide.* 004-2178-005.

[7] Cray Inc. *Programming Environment Release Overview.* S-5212-35.

[8] O-J. Dahl, B. Myrhaug, and K. Nygaard. Simula common base language. Technical Report S22, Norwegian Computing Center, Oslo, 1970.

[9] Margaret Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual.* Addison-Wesley, 1990.

[10] Boris Fomitchev. STLport story. Technical report, STLport Consulting Inc., 2001. `http://www.stlport.org/doc/story.html`.

[11] International Standards Organization (ISO). *Programming Languages — C*, 1990. ISO/IEC 9899:1990.

Core Language		
C++ Compiler	Cray C++	KAI C++
Release	3.5	3.4*
Bool	++++	++++
Covariant return	−−	++++
Exceptions	++++	++++
Explicit	++++	++++
Export	−−	−−
For scope	++++	++++
Koenig lookup	++++	++++
Namespaces	++++	++++
New throws		++++
RTTI	++++	++++
Template basics	++++	++++
Template as parameter	−−	−−
T/C partial specialisation	++++	++++
T/F explicit arguments	++++	++++
T/F partial ordering	++++	++++
T/F members	++++	++++
Typename	++++	++++

Standard Library			
Library	CrayLibs	STLport	KCC
Release	3.5	4.0	3.4*
Auto_ptr	−−	++++	++++
Exception hierarchy	−−	++++	++++
Iostream templates	−−	−−	++++
Locales	−−	−−	++
Namespace std	−−	++	++++
Numeric_limits	−−	++++	++++
STL algorithms	++	++++	++++
STL containers	++	++++	++++
STL iterators	++	++++	++++
String class	++	++++	++++
String streams	−−	−−	++++
String template	++	++++	++++
Valarray	−−	++++	++++

Scoring	
−−	The feature is not present
+	A token attempt at the feature is made
++	Basic support of the feature is present and usable
+++	The feature is well supported
++++	The feature seems to be completely standard compliant

*Evaluation for KCC taken from [29]

Table 1: C++ compiler and library comparison chart

[12] International Standards Organization (ISO). *Programming Languages — C++*, 1998. ISO/IEC 14882:1998.

[13] Thomas Jell and Axel von Reeken. *Objektorientiertes Programmieren mit C++: eine Einführung mit vielen Beispielen, Übungsaufgaben und Musterlösungen*. Hanser, 2nd edition, 1993.

[14] Nicolai Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Addison-Wesley, 1999.

[15] Brian Kernighan and Dennis Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[16] Brian Kernighan and Dennis Ritchie. *The C Programming Language. ANSI C*. Prentice-Hall, 2nd edition, 1988.

[17] Kuck & Associates. *KAI C++ Compiler (KCC) Release Notes for Cray T3E*, 1998.

[18] Kuck & Associates. *KAI C++ Compiler (KCC) User's Guide*, 1998.

[19] Kuck & Associates. *KAI C++ Standard Compliance*, 2001. `http://www.kai.com/C_plus_plus/Current/doc/standard.html`.

[20] Angelika Langer and Klaus Kreft. *Standard C++ IOStreams and Locales: Advanced Programmer's Guide and Reference*. Addison-Wesley, 2000.

[21] Stanley Lippmann. *C++ Primer*. Addison-Wesley, 1989.

[22] Modena Software, Inc. *Modena C++ Standard Library*, 1997. `http://www.modena.com/libpp.htm`.

[23] David Musser and Atul Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley, 1996.

[24] P.J. Plauger. *The Draft Standard C++ Library*. Prentice-Hall, 1995.

[25] Thomas Plum and Danial Saks. *C++ Programming Guidelines*. Plum Hall, 1991.

[26] Peter Prinz and Ulla Kirch-Prinz. *C++. Lernen und professionell anwenden*. MITP, 1999.

[27] Peter Prinz and Ulla Kirch-Prinz. *Objektorientiert programmieren mit ANSI C++*. Markt und Technik, 1999.

[28] SGI. *Standard Template Library Programmer's Guide*, 1996.

[29] Ross Smith. C++ compiler comparison chart. Technical report, The Internet Group, 2000. `http://animal.ihug.co.nz/c++/compilers.html`.

[30] Alexander Stepanov and Meng Lee. The standard template library. Technical Report HPL-94-34, HP Labs, August 1994.

[31] STLport Consulting Inc. *STLport License Agreement*, 2001. `http://www.stlport.org/doc/license.html`.

[32] STLport Consulting Inc. *STLport Release Notes 4.0*, 2001. `http://www.stlport.org/doc/release_notes.html`.

[33] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

[34] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley, 1994.

[35] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 4th edition, 2000.

[36] Gregory V. Wilson and Paul Lu, editors. *Parallel Programming Using C++*. Scientific and Engineering Computation. MIT Press, 1996.

[37] P.M. Woodward and S.G. Bond. *Algol 68-R Users Guide*. Her Majesty's Stationery Office, London, 1974.

[38] Daoqui Yang. *C++ and Object-Oriented Numeric Computing for Scientists and Engineers*. Springer, 2001.

About the authors

Hans-Hermann Frese is Chairperson of the CUG Programming Environment Group. He received a diploma in mathematics from the Universität Bielefeld, Germany, and works as a consultant for computer science at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). His research interests include high-performance computing systems, new architectures and programming environments, and the evaluation and benchmarking of HPC systems.

Detlef Reichardt received his Ph.D. in theoretical physics in 1990. He has done research in several scientific areas like molecular dynamics, quantum chemistry, and high-performance computing on massively parallel systems. After leaving Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) where he worked as a consultant for computational chemistry he is now a self-employed instructor for computer science.

Philip Rohwetter studies physics at the Freie Universität Berlin. He also works for the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB).