ANNUAL REPORT FOR AWARD # 0131354

Andrew Lumsdaine ; *Indiana University*
NGS: Open Compilation for Self-Optimizing Generic Components

Participant Individuals:
Senior personnel(s) : David Musser; Sibylle Schupp
Graduate student(s) : Douglas Gregor; Brian Osman; Jeremy Siek; Lie-Quan Lee
Post-doc(s) : Jaakko Jarvi
Graduate student(s) : Ronald Garcia; Michael L LaSpina; Mayuresh Kulkarni
Undergraduate student(s) : Kyle Ross
Graduate student(s) : Jeremiah J Willcock

Participants' Detail

Partner Organizations:
Rensselaer Polytechnic Institute: Collaborative Research; Personnel Exchanges

Edison Design Group: In-kind Support

Code Sourcery, LLC: In-kind Support; Collaborative Research

Other collaborators:

Shin-Ming Liu, Hewlett-Packard

Activities and findings:

Training and Development:

This project has involved several grad students and one post-doc, all of whom have received research experience as a result.

This project has given these individuals some unique experiences because of its cross-cutting nature. That is, the research personnel have received exposure to compiler research, software library research, formal language theory, as well as exposure to particular problem domains (e.g, numerical linear algebra).

Finally, this project has emphasized the importance of relevance and the research personnel have been cognizant of that issue and have gained experience with real production compilers.

Outreach Activities:

We have already organized one outreach activity and are in the process of organzing another:

1) We hosted a workshop for the Midwest Society on Programming Languages and Systems at IU Bloomington on April 13, 2002.

2) We are organizing a workshop on Software Library Implementation, Design, and Evaluation (SLIDE) and hope to co-locate it with PLDI'03.

The software libraries that have been developed as part of this project are intended for public consumption and have been released through the highly-visible Boost Library Group (www.boost.org). Some parts of these (and other) libraries will be proposed for incorporation into the C++ Standard Library.

Our compiler work is similarly being done in the context of production compilation systems (gcc and EDG) with the expectation that the results of our work can be readily adopted by real programmers.

Finally, much of the work that has been done with respect to generic programming and compiler development has direct impact to the C++ programming language itself (and to the libraries that are part of the language standard). Several of the participants in this project are active in the ISO/IEC standards committee for the programming language C++ and have made important contributions to the standard library and to the language itself.

Journal Publications:

D. Gregor, S. Schupp, D. Musser, "Design Patterns for Library Optimizations", *Scientific Programming*, vol. , (), p. . Accepted

D. Gregor, S. Schupp, "Making the Usage of STL Safe", *IFIP Working Conference on Generic Programming.*, vol. , (), p. . Accepted

S. Schupp, D. P. Gregor, B. Osman, D. R. Musser, J. G. Siek, L.-Q. Lee, A. Lumsdaine, "Concept-Based Component Libraries and Optimizing Compilers", *IPDPS'02*, vol. , (2002), p. 1. Published

Ronald Garcia, Jaakko Jarvi, Andrew Lumsdaine, Jeremy G. Siek, and Jeremiah Willcock , "A Comparative Study of Language Support for Generic Programming", *Proceedings of 2003 ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA'03)*, vol. , (), p. . Accepted

Jaakko Jarvi, Jeremiah Willcock, Howard Hinnant, and Andrew Lumsdaine, "Function Overloading Based on Arbitrary Properties of Types", *C/C++ Users Journal*, vol. 21(6), (2003), p. 25. Published

Jaakko Jarvi, Jeremiah Willcock, and Andrew Lumsdaine, "Concept Controlled Polymorphism", *GPCE'03*, vol. , (), p. . Submitted

S. Schupp and D. R. Musser and D. P. Gregor and S.-M. Liu, "Semantic and Behavioral Library Transformations", *Information and Software Technology*, vol. 44, (2002), p. 797. Published

D. R. Musser and Z. Shao, "Concept Use or Concept Refinement: An Important Distinction in Building Generic Specifications", *Proceedings 4th International Conference on Formal Engineering Methods (LNCS)*, vol. 2495, (2002), p. 132. Published

W. Klostermeyer and D. R. Musser and A. J. Sanchez-Ruiz, "Complete Traversals as General Iteration Patterns", *Generic Programming, IFIP TC2/WG2.1 Working Conference on Generic Programming*, vol. 243, (2003), p. . Accepted

Book(s) of other one-time publications(s):

Jaakko Jarvi, Andrew Lumsdaine, and David S. Wise, "MSPLS 2002: Proceedings of the Workshop of the Midwest Society for Programming Languages and Systems" , bibl. Technical Report 560, Indiana University Computer Science, (2002). *Workshop Proceedings* Published
of Collection: Jaakko Jarvi, Andrew Lumsdaine, and David S. Wise, "MSPLS 2002: Proceedings of the Workshop of the Midwest Society for Programming Languages and Systems"

Other Specific Products:

Software (or netware)

```
ISO/IEC proposal N1478 03-0061

Jaakko Jarvi, Bjarne Stroustrup, Douglas Gregor, and Jeremy Siek:
Decltype and auto : Programming language C++. Technical Report
N1478=03-0061, ISO/IEC JTC 1, Information technology, Subcommittee SC
22, 2002.

This proposal is being considered for inclusion in the ISO/IEC
standard for the programming language C++.
```

Software (or netware)

```
ISO/IEC proposal N1403 02-0061

Jaakko Jarvi: Proposal for adding tuple types into the standard
library : Programming language C++. Technical Report N1403=02-0061,
ISO/IEC JTC 1, Information technology, Subcommittee SC 22, 2002.

This proposal was accepted into the Library Technical Report for the
C++ programming language standard.
```

Software (or netware)

```
ISO/IEC Proposal N1455 03-0038
```

Peter Dimov, Douglas Gregor, Jaakko Jarvi, and Gary Powell: A proposal to add an enhanced binder to the library technical report. Technical Report N1455=03-0038, ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal was accepted into the Library Technical Report for the C++ programming language standard.

Software (or netware)

ISO/IEC Proposal N1483 03-0066

Douglas Gregor, Gary Powell, and Jaakko Jarvi: Typesafe variable-length function and template argument lists. Technical Report N1483=03-0066, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal is under consideration for inclusion in the Library Technical Report for the C++ programming language standard.

Software (or netware)

ISO/IEC Proposal N1476 03-0059

David Abrahams, Jeremy Siek, Thomas Witt: Iterator Facade and Adaptor. Technical Report N1476=03-0059, ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal is under consideration for inclusion in the Library Technical Report for the C++ programming language standard.

Software (or netware)

ISO/IEC Proposal N1477 03-0060

David Abrahams, Jeremy Siek, Thomas Witt: New Iterator Concepts. Technical Report N1477=03-0060, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal is under consideration for inclusion in the Library Technical Report for the C++ programming language standard.

Software (or netware)

ISO/IEC Proposal N1453 03-0036

Douglas Gregor. A proposal to add a reference wrapper to the standard library. Technical Report N1453=03-0036, ISO/IEC JTC 1, Information

technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal was accepted into the Library Technical Report for the C++ programming language standard.

Software (or netware)

ISO/IEC Proposal N1454 03-0037

Douglas Gregor. A uniform method for computing funciton object return types. Technical Report N1454=03-0037, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal was accepted into the Library Technical Report for the C++ programming language standard.

Software (or netware)

ISO/IEC Proposal N1455 03-0038

Douglas Gregor. A proposal to add an enhanced binder. Technical Report N1453=03-0036, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming language C++, 2003.

This proposal was accepted into the Library Technical Report for the C++ programming language standard.

Software (or netware)

The Boost MultiArray Library: A generic N-dimensional array concept definition and common implementations of that interface.

http://www.boost.org/libs/multi_array/doc/index.html

Software (or netware)

The Boost Lambda Library: Lambda abstractions for C++

http://www.boost.org/libs/lambda/doc/index.html

Software (or netware)

Boost Dynamic Bitset Library

http://www.boost.org/libs/dynamic_bitset/dynamic_bitset.html

Internet Dissemination:

http://www.cs.rpi.edu/research/gpg/Simplicissimus http://www.cs.rpi.edu/~musser/gp/algorithm-concepts

Contributions to Resources for Science and Technology:

Support for the material in the textbook Accelerated C++ and the Algorithm Concepts taxonomy (both of which are discussed in the Research and Education Activities section) are significant resources for research and education. The Algorithm Concepts taxonomy has already been in use in for the past two years in the Computer Algorithms (upper-level undergraduate) course at Rensselaer; in fact, there was considerable student participation in the development of these web documents as course projects.

Contributions Beyond Science and Engineering:

Work in this project has been done with respect to generic programming and compiler development has direct impact to the C++ programming language itself (and to the libraries that are part of the language standard). Several of the participants in this project are active in the ISO/IEC standards committee for the programming language C++ and have made important contributions to the standard library and to the language itself. These contributions will impact all programmers using the C++ programming language.

Special Requirements for Annual Project Report:

Unobligated funds: less than 20 percent of current funds

Categories for which nothing is reported:

Research and Education Activities
Findings
Contributions Within Discipline
Contributions to Other Disciplines
Contributions to Education and Human Resources
Special Reporting Requirements
Animal, Human Subjects, Biohazards

Submit Return

View Activities PDF File

View Findings PDF File

Research and Education Activities

This project advances the state of the art in high-performance software, specifically related to the design and compilation of high-performance scientific libraries. This work addresses two of the most important issues—code complexity and performance optimization—by applying and extending two emerging programming paradigms: generic programming and concept-based optimization. The vehicle for this work is the development of an integrated framework consisting of open compilation and generic library technologies. By simultaneously developing the compilation technologies and the library technologies—and by providing mechanisms to couple the two together—the whole will be greater than the sum of its parts.

Generic Programming. We are developing and extending design methodologies for generic programming and applying these results to develop a component taxonomy for target library problem and solution domains.

We conducted a comprehensive comparison of generics in six programming languages: C++, Standard ML, Haskell, Eiffel, Java (with its proposed generics extension), and Generic C#. For the study, a substantial example of generic programming was implemented in each of these languages, with the goal of identifying important language features that support generic programming. We identified eight such featuers, and claim that these features are necessary to avoid awkward designs, poor maintainability, unnecessary run-time checks, and painfully verbose code. As languages increasingly support generics, it is important that language designers understand the features necessary to provide powerful generics and that their absence causes serious difficulties for programmers.

We have continued to refine the Algorithms Concept taxonomy whose development was begun in the first year, adding concept descriptions of additional sequence algorithms in STL and graph algorithms in the Boost Graph Library. The current algorithms concept web is now publicly available at `http://www.cs.rpi.edu/~musser/gp/algorithm-concepts`. Progress has been made also on identifying and formally characterizing an initial set of distributed algorithm concepts to be included in the concept web as part of this project.

Open Compilation. The open compiler environment is presently being implemented as an extension of the gcc compiler and the EDG C++ front-end. This framework will be used to develop a self-optimizing library consisting of generic components having the performance and flexibility required of complex scientific applications. We have also worked with CodeSourcery to develop a portable and extensible representation of abstract syntax trees for C++ programs. This representation will facilitate a component-based approach to implementation of our open compilation framework.

Concept-based Optimization. We have prototyped a concept-based simplifier to allow simplification transformations to be applied to higher-order (user-defined) types. Extensions to this work will incorporate other important transformations (as well as algorithm concepts) into the concept-based optimzation process.

Library Development. We are currently developing several generic libraries, in the areas of multithreading, distributed computing, multi-dimensional arrays, and parallel numerical linear algebra. Current work is focusing on formalizing the interfaces for these libraries and incorporating concept-based optimization techniques.

Several libraries developed as part of this project have been widely distributed on-line through the Boost Library Group. Some parts of these (and other) libraries developed by our group have been proposed for incorporation into the C++ Standard Library.

Tool Development. More progress has been made towards the development of a tool for the safe use of generic library components. As we had come to realize earlier, it is important for the acceptance of generic libraries to offer the same kind of support that non-generic code enjoys. We had therefore started in year one of this project to extend the ordinary semantic analysis of a typical compiler with checks for the correct composition of library components. This extension has now grown into a stand-alone tool, STLlint, a Lint-like checker for generic libraries in the spirit of STL.

Within STLlint, our main focus has been on loops with monotonic induction variables, since the most important idiom of generic libraries is the traversal of linear sequences via iterators. On the theoretical side, we have worked on improving the control-flow merge behavior with respect to integral relations among object fields (e.g., to capture in a sufficiently precise, yet feasible manner the relations between an iterator and the container it is accessing). We are preparing a journal article that summarizes our findings.

Benchmark programs for testing STLlint are being taken from the textbook *Accelerated C++* (by Andrew Koenig and Barbara Moo), a widely used introductory text. By applying STLlint to the examples and exercises of this text (and varying the input variables accordingly) we are able to test STLlint both with a subset of C++ that is used in practice and with realistic errors that novice users make. Our ultimate goal is to integrate STLlint into introductory courses that use C++. Currently, we can check about 50% of the material in the *Accelerated C++* textbook.

Findings

Simplicissimus. The implementation of the first version of the expression optimizer (”Simplicissimus”) has been completed, integrated into the GNU compiler, and made available on the World Wide Web. Our framework was tested using different generic libraries, in particular our Matrix Template Library (MTL) as well as with application libraries on top of and combined with MTL. During these tests, we discovered that the rewrite-techniques underlying the expression optimizer can also be used to overcome interoperability problems of libraries due to conflicting design principles of the libraries involved. We therefore conducted a series of experiments, in which we demonstrated how our tool can be used to transform libraries in a systematic, semantics-based, and safe manner, that helps to avoid the overhead in time or space that might occur when libraries are plugged together. The results of these experiments are published in an article that has been accepted for the IST journal.

Generic Libraries. A second insight is related to the usage of generic libraries. As advantageous as their flexibility and thorough parameterization is from the standpoint of reusability, readability, and maintenance, our experience has shown that the usage of generic libraries comes with a unique set of problems that are not adequately addressed by current compilers or development tool technology. We therefore started extending the phase of semantic analysis by checks for the correct composition of library components. We have already modified the EDG front end so that it is possible to incorporate error messages at the component level. We are currently investigating how to adjust the various phases of traditional static analysis, most notably loop analysis, so that they can recognize and analyze library components. In addition to improving the support for users, this type of high-level analysis will also be essential in moving from the currently expression-based optimizations to more powerful optimizations at the statement level.

Formal Description of Concepts. In order to provide better language support for generic programming, we have developed a formalism for describing concepts and concept hierarchies. In our formalism, concepts are defined as type functions mapping function names to function types. Definitions for concept refinement and modeling by types follow readily.

Language Support for Generic Programming. Many modern programming languages support basic generic programming, sufficient to implement type-safe polymorphic containers. Some languages have moved beyond this basic support to a broader, more powerful interpretation of generic programming, and their extensions have proven valuable in practice. We conducted a a comprehensive comparison of generics in six programming languages: C++, Standard ML, Haskell, Eiffel, Java (with its proposed generics extension), and Generic C#. By implementing a substantial example in each of these languages, we identified eight language features that support this broader view of generic programming. We found these features are necessary to avoid awkward designs, poor maintainability, unnecessary run-time checks, and painfully verbose code. As languages increasingly support generics, it is important that language designers understand the features necessary to provide powerful generics and that their absence causes serious difficulties for programmers. Our complete findings are reported in “A Comparative Study of Language Support for Generic Programming,” to appear in the Proceedings of OOPSLA’03.