GGCC
(ITEA ~ 05012)

Arnaud Laprévote, Mandriva, France

SMEs show the way to quality GCC compilation

A consortium with an unusual predominance of SME partners has added a new branch to the widely-used global GNU Compiler Collection (GCC). The tools developed in the GGCC project allow the optimisation of complete programs and libraries – including static analysis for early bug fixes. This could help Europe profit from a range of interesting commercial opportunities.

Explosive growth in the software content of digital systems, coupled with increasing expectations that operation will be trouble-free, creates overwhelming problems for program developers. Integration of components and validation of complete packages now consumes 70 to 90% of total development costs.

Large corporations spend heavily on proprietary systems for the most mission-critical applications in sectors such as aerospace. But even here, the investment required to achieve zero-defect quality can prove ruinous. Most providers of enterprise, consumer and embedded software cannot even afford the same levels of expenditure. Yet, while quality assurance remains a heavy burden, cutting corners may lead to brand-damaging numbers of faults in shipped products.

BUILDING ON INTERNATIONAL STANDARD

Open-source tools are increasingly employed to resolve this dilemma. GCC, for example, has become the standard compiler for most modern Unix-like computer operating systems, including GNU/Linux, the BSD family and Mac OS X. It has been adapted to a wide variety of processor architectures, and is widely deployed as a tool in commercial, proprietary and closed-source software development environments.

GCC is also available for most embedded platforms, including those for top-name gaming consoles. Several companies base their businesses on supplying and supporting GCC ports to such platforms, and today's chip manufacturers consider its availability almost essential to the success of an architecture.

This highly portable compiler suite offers exceptional hardware- and vendor-independence as it is able to generate code for almost every current instruction set architecture. With a massive user base and contributions from hundreds of professional developers, mainly in the USA, its content has grown tenfold over the past decade.

The principal advantages of GCC are code generation quality and compilation speed. However, despite its huge expansion, an aspect that long remained unexplored was global static analysis and coding-rule checking for debugging and project-wide optimisation. Global analysis makes it possible to process complete compilation units such as programs or libraries together, and to manage the relationships between them.

Coding-rule validation ensures programs conform to rules established for a particular industry, company or application. Cumulatively, they further improve coding efficiency, enhance program performance and reduce testing times – cutting costs and time to market.

Earlier work focused on academic prototypes or closed commercial products such as Klocwork and PolySpace. GGCC broke new ground by incorporating coding-rule validation into a popular mainstream platform as a basis for new tools and interfaces applicable across many industries.

REINFORCING COMPETITIVENESS

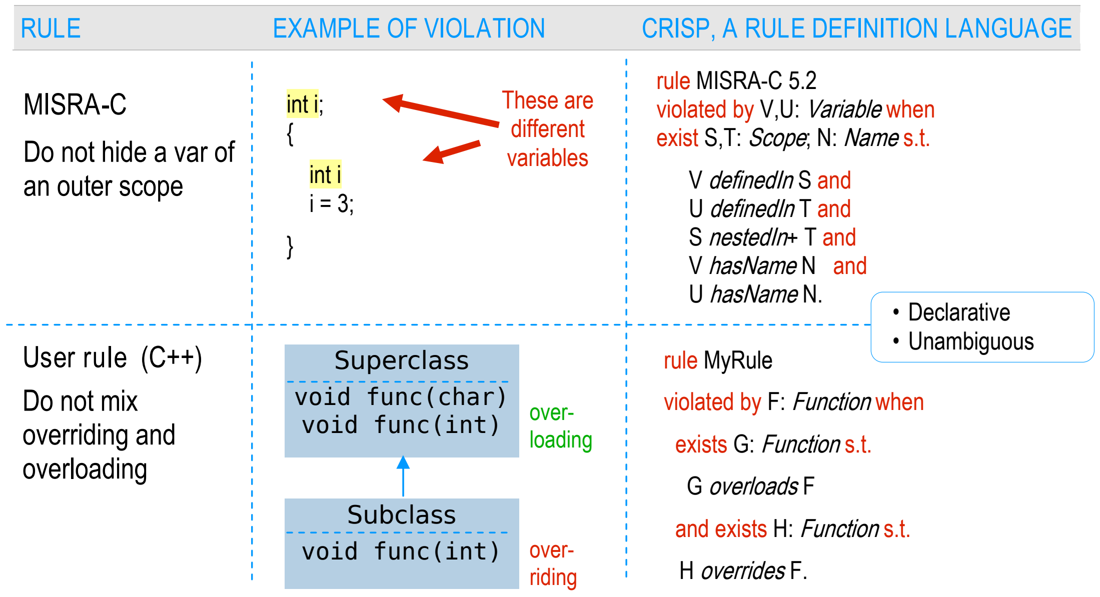
One of the direct aims of the ITEA project was to reinforce the competitiveness of computer software providers. By improving the quality of the open-source C/C++/Fortran/Java compiler and extending it with static analysis, development time would be shortened and test effort diminished – thus reducing time to market.

Software-hazards detection exposes potential issues in source code, hence lowering costly testing efforts; coding rules conformance makes corporate knowledge on specific software development processes explicit, and its automatic validation helps enhance software quality. Furthermore, GGCC integrates the legacy compilation functionalities but also provides quality assurance and verification & validation (V&V) functionalities in one global package.

And finally, global optimisation opens GGCC to new markets, allowing cutting-edge program optimisation to be integrated into production environments. Target-specific compilers often produce up to 50% faster code than GCC on real applications running on general-purpose processors such as Pentium or PowerPC. The gap can grow to factors of three to five on specific or embedded architectures like Philips Trimedia or Intel Itanium.

Improved performance is important for several reasons, not the least of which is that code that uses machine resources more efficiently can be executed

CODING RULE CHECKING


RULE
EXAMPLE OF VIOLATION
CRISP, A RULE DEFINITION LANGUAGE
MISRA-C
Do not hide a var of an outer scope
These are different variables
Declarative
Unambiguous
User rule (C++)
Do not mix overriding and overloading
Superclass
Subclass
over-loading
over-riding

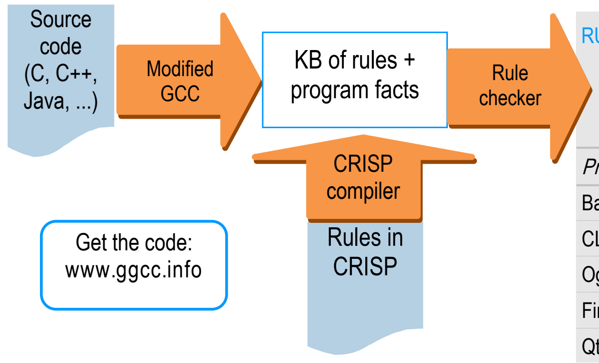
```
int i;
{
  int i
  i = 3;
}
```

rule MISRA-C 5.2
violated by V,U: *Variable* when
exist S,T: *Scope*; N: *Name* s.t.
  V *definedIn* S and
  U *definedIn* T and
  S *nestedIn+* T and
  V *hasName* N and
  U *hasName* N.

```
Superclass
void func(char)
void func(int)

Subclass
void func(int)
```

rule MyRule
violated by F: *Function* when
 exists G: *Function* s.t.
  G *overloads* F
 and exists H: *Function* s.t.
  H *overrides* F.


Source code (C, C++, Java, ...)
Modified GCC
KB of rules + program facts
Rule checker
CRISP compiler
Rules in CRISP
Get the code: www.ggcc.info

RULE VIOLATIONS REPORT

Some High-Integrity C++ rules implemented and tested with open-source projects:

Project	*Violations*	*Time (s)*
Bacula	3	0.03
CLAM	131	0.84
Ogre	94	1.46
Firebird	191	1.40
Qt	1249	13.06

using slower hardware. Such hardware is typically cheaper and/or less power hungry, both extremely important aspects for embedded systems.

NOTABLE INNOVATIONS

GCC's size, complexity and dynamic nature, plus the strict rules imposed by the GCC community, made the necessary development work daunting. The consortium nevertheless achieved significant progress, developing a common framework to accommodate three main functions: analysis and optimisation, hazard detection and code validation.

It was necessary to integrate a higher level language as GCC is written in C, which – being independent of the system in which it is implemented – is not ideal for abstract interpretation. GGCC opted for Lisp and developed a middle-end Lisp-to-C translator (MELT), which has been adopted as an official branch of the GCC development tree. MELT can be used to detect parallelism in code and has many potential applications in prototyping. Within the project, the first simple static analysers were realised using MELT.

A mature version of the global optimisation framework, bringing greatly enhanced performance and tighter integration with the GCC internal application program interfaces was also developed. Even if still very slow, the global optimisation framework has shown very good performances on par with proprietary compilers on the test codes that were chosen.

Add-ons include coding rules in sugared Prolog (CRISP), a logic-based domain-specific language making it possible to express coding rules easily even without relevant expertise.

GGCC successfully tested the prototype coding rule checker at full scale on Nokia's C++ Qt cross-platform library. The number of rules remains limited, but the results offer a promising indication of code quality.

TWO-WAY BENEFIT

GGCC was unusual in several respects. Most ITEA consortia are led by large companies, whereas this project was SME-led. For a number of the SME members, it was also a first venture into transnational co-operation. This created some operational difficulties at the start, but the experience has proved valuable – and several partners are already aiming to take part in further collaborations.

Moreover, while SME partners were very interested in software quality, none was a professional compiler specialist. The initial goal was to produce tools for internal use, taking the view that there is a real need for effective quality-enhancement tools because the cost of bug correction is huge. If the project outcomes allowed automatic solution of even 10% of the problems, the effort could be considered worthwhile.

Although the results at the end of the funded period remained at a relatively early stage of refinement, they were nevertheless sufficiently advanced to be suitable for offer to other projects in the field.

Many of the improvements have already helped GCC to become a leading platform for global optimisation projects that go way beyond the capabilities of competing compilers. By acquiring leading-edge expertise, the partners have created an o pportunity for Europe to establish a strong position in an area with considerable potential.