
GCC: Looking Forward

Mark Mitchell

CodeSourcery, LLC

GCC Summit 2003

First Thoughts

- Acknowledgements
- How I came to be giving this talk.
- Disclaimers
 - This is not a scientific talk.
 - I am not an objective observer.
 - Proposals are initial concepts.

Outline



Current Status

- History
- Possible Improvements:
 - Process changes.
 - Bigger ideas.
- Questions and Answers
- Party!

GCC Successes

- Standard compiler on:
 - GNU/Linux
 - Other free operating systems
 - OS X
- Used on almost every other platform:
 - Windows
 - Embedded systems
 - UNIX and UNIX-like systems

We all deserve a pat on the back.

Why is GCC Successful?

- Support for a very wide variety of architectures.
 - Same behavior on all systems.
 - Support for kernel/embedded programming.
- Good code generation.
- Standards compliance.
- The price is right.

Technical Weaknesses?

- Run-time performance.
 - Difficult to measure.
 - Varies from architecture to architecture.
 - It is easier to implement algorithms to tune them.
- Compile-time performance.
 - Difficult to measure.
 - Varies from architecture to architecture.
 - Multiple small losses result in large losses.
 - Additional optimizations slow the compiler.

Outline

- Current Status
- History
- Possible Improvements:
 - Process changes.
 - Bigger ideas.
- Questions and Answers
- Party!

GCC: The Early Years

- Private FTP site.
- Small group of developers.
- A single person directed development.

Frustration with this model lead to EGCS.

GCC: The Present

- Public CVS repository.
- Large number of developers.
- GCC Steering Committee resolves issues.

A big improvement!

Development Process

- A volunteer feels like doing something.
- The volunteer writes code, often in relative isolation.
- The code is posted to gcc-patches.
- It's sometimes reviewed.
- It gets checked in.

Regressions

- Determining who caused a problem is difficult.
 - Latent defects.
 - Defects not noticed immediately.
- Developers do not always correct the problems.
 - Difficult reproducing problems.
 - Problems arise outside the area of expertise of the developer.
 - The developer is busy.
- Ultimately, nobody is responsible for correcting problems.
 - Volunteers perform many useful tasks, but cannot be expected to solve everything.
 - Even maintainers are volunteers.

Testing

- Large regression testsuite
 - But only tests the platform the developer is using.
- Some automated regression testers
 - But only for some platforms, and often slow to pinpoint all regressions.
- Irregular use of proprietary testsuites
 - Plum-Hall, Perennial, ACATS
 - CodeSourcery C++ ABI, AltiVec, etc.
- No unit testing
 - Optimization regressions often undetected.
 - No easy method for validating internal algorithms.

Interfaces

- What interfaces?
 - It would take work to define them.
 - To some extent, we avoid them on purpose.
- Problems:
 - Scope of change is unpredictable.
 - Steep learning curve.
 - All code ends up in the tree.
 - Releases have to be coordinated.

No Development Plan

- There is no development roadmap:
 - What features are going to be in GCC X.Y?
 - Who will implement them?
 - What if those changes cause new defects?

Who is responsible?

Proprietary Software

- Owner minimizes long-term costs:
 - Invest in architecture.
 - Re-architect as necessary.
 - Build code that is easy to maintain as possible.
- Competitive advantage:
 - Higher-quality product.
 - Lower cost structure.

Free Software

- Contractor minimizes short-term costs:
 - Make minimal changes required to satisfy customer.
- Competitive advantage:
 - Superior knowledge of code.
 - Superior development ability.
 - Semi-proprietary additions.
 - Code understood only by developer.
 - Code maintained outside of public source base.

Bad Incentives

- Typical GCC contracts:
 - Provide port to platform P for \$X
 - Implement feature F for \$X
- Contractor's objectives:
 - Minimize time to delivery.
 - Minimize costs.
- Problems:
 - Documentation provided by contractor helps contractor's competitors.
 - A more general implementation may be both more expensive and more helpful to customer's competitors.
 - Communicating with competitors is dangerous.

Winning GNU/Linux Strategies

- **Low-cost operating system:**
 - Attractive to users.
 - Reduced costs for hardware vendors.
 - Did not reduce revenues for hardware vendors:
 - Operating systems and compilers are loss-leaders
- **High-quality operating system:**
 - Reliable.
 - POSIX standard enhances portability, reduces competition.
 - Open source strategy permitted customization:
 - Ports
 - Super-computing, highly-embedded systems, etc.

GCC vs. GNU/Linux

- GCC is low-cost
 - But defects are more difficult for users to understand and assess.
- Compilers are a loss-leader
 - But microprocessor vendors sometimes consider compiler algorithms highly proprietary.
- Quality is high
 - But neither front-ends nor back-ends are as good as proprietary vendors.

Goals for GCC?

- Higher-quality product.
 - Fewer defects.
- Code-generation strategy.
 - GCC-compatible compilers exist, and generate better code.
- Infrastructure improvement.
- Better incentives for market participants.
- Make GCC the GNU/Linux of compilers.

Outline

- Current Status
- History
- Possible Improvements:
 - Process changes.
 - Bigger ideas.
- Questions and Answers
- Party!

Proposed Development Process

- Identification of need
 - Example: Poor code generation on platform X
- Written proposal
 - Example: Provide a new scheduler.
- Technical design
 - Example: Use Smith-Jones algorithm.
 - Includes validation plan.
- Implementation
 - Example: Write code.
- Validation
 - Example: Set of benchmarks to run.

Examples

Other groups use similar methods:

- Python PEPs:
 - Formal proposals for new features to be added to Python.
- Apache Long-Term Plans
- ISO C/C++ Committees
- Proprietary compiler developers
- Free compiler developers – but only internally.

Volunteerism

- Volunteers are incredibly valuable.
 - Example: Wolfgang Bangerth reviews PRs.
- Volunteers become major contributors.
 - Example: Nearly all of us.

But major contributions to GCC tend to be commercial.

Commercial Work

- New scheduler.
- New ports.
- New C++ parser.
- Pre-compiled headers.
- Release management
- ...

We need to balance volunteerism and commercialism.

Compilers and Hardware Sales

- Performance-conscious consumers consider the combined performance of:
 - Hardware
 - Operating System
 - Networking, threading, I/O, etc..
 - Software
 - Web servers, databases, etc.
- Some consumers are not performance-conscious:
 - Uptime and reliability, network performance

Compilers are necessary, but they are not core technology.

Compiler Development is Costly

- Development costs:
 - Millions of lines of code.
 - Ten-plus full-time developers:
 - Millions of dollars annually
- Support costs:
 - Support personnel
 - Non-standard behaviors impede porting.

GCC: A Common Need

- Shared development cost.
- Cross-platform standard.
- Users demand GCC.
- Open-source is insurance.

Vendors can win by working together.

A GCC Consortium?

- Not-for-profit corporation
- Industrial and governmental partners
 - Membership dues
 - Voting rights
- Objectives:
 - Checks and balances
 - Long-term planning
 - Funding through competitive bids

Funding Objectives.

- Improved run-time and compile-time performance
 - Modern algorithms.
 - Re-architecture as necessary.
- Documentation of design and implementation.
 - Formal specification of internal representation.
- Development of regression tests.
 - Improved modularity to permit unit-testing.
- Monitoring of compiler quality.
- Processing of submitted patches.

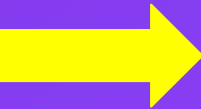
Summary

- A more structured development process:
 - Proposals, design documents, validation.
- A GCC consortium:
 - Support the development process.
 - Funding for long-term projects.

Outline

- Current Status
- History
- Possible Improvements:
 - Process changes.
 - Bigger ideas.
- Questions and Answers
 - Party!

Outline

- Current Status
 - History
 - Possible Improvements:
 - Process changes.
 - Bigger ideas.
 - Questions and Answers
-  Party!

GCC: Looking Forward

Mark Mitchell

CodeSourcery, LLC

GCC Summit 2003