GNU Compiler Collection

Alcino Dall'Igna Júnior

Instituto de Computação
Universidade Federal de Alagoas

18 de agosto de 2006

Roteiro

Roteiro

O que é hoje

Versões

- 24 Maio 2006: GCC 4.1.1
- 10 Mar 2006: GCC 4.0.3
- 06 Mar 2006: GCC 3.4.6
- 28 Fev 2006: GCC 4.1.0

Linguagens suportadas

- C (gcc) e C++ (g++)
- Objective-C e Objective-C++
- Fortran 95 (gfortran)
- Java (gcj)
- Ada (gnat)
- não oficialmente: Pascal, D, Modula-2, Modula-3, Mercury, VHDL e PL/I

O que é hoje

Versões suportadas pelo GCC 4.1

- C - gcc
  - suporte completo para ISO/IEC 9899:1990 (C89), incluindo as correções de 1994 e 1996
  - suporte quase completo para ISO/IEC 9899:1999 (C99), incluindo correções de 2001 e 2004 (http://gcc.gnu.org/gcc-4.1/c99status.html); ainda não suporta VLA (*variable length array*)
  - default: -std=gnu89, C99 com extensões GNU, que deverá passar para -std=gnu99
- C++ - g++
  - Não documentado o quanto g++ está aderindo a0 ISO/IEC 14882:1998 ou 2003

O que é hoje

Versões suportadas pelo GCC 4.1

- Fortran - gfortran
  - suporte a ISO/IEC 1539:1997 (Fortran95)
  - previsto suporte para outras versões (77, 90 e 2003)
- Java - gjc
  - suporta quase integralmente java 1.2, e parte da versão 1.4
  - aprovada migração para Java Eclipse, com suporte para Java 1.5
  - característica interessante: compila fonte para bytecode e para código nativo, bem como bytecode para código nativo
- Ada - gnat
  - implementa basicamente Ada95

O que é hoje

Arquiteturas suportadas

- Alpha, ARM, Blackfin, H8/300
- IA-32 (x86), AMD64 (x86-64), IA-64 (ItaniC)
- IBM System/370, System/390
- MIPS, Motorola 68000 e 88000
- PA-RISC, PDP-11, PowerPC
- SuperH, SPARC, VAX
- famílias Renesas R8C/M16C/M32C
- família MorphoSys
- Outras arquiteturas menos conhecidas: A29K, ARC, Atmel AVR, C4x, CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, NS32K, ROMP, Stormy16, V850, and Xtensa.
- Suportados separadamente: D10V, PDP-10, MicroBlaze, MSP430 e Z8000

O que é hoje

Comitê Gestor - GCC Steering Comittee

- Per Bothner
- Joe Buck (Synopsys)
- David Edelsohn (IBM)
- Kaveh R. Ghazi
- Jeffrey A. Law (Red Hat)
- Marc Lehmann (TU Karlsruhe)
- Jason Merrill (Red Hat)
- David Miller (Red Hat)
- Mark Mitchell (CodeSourcery)
- Toon Moene (Koninklijk NMI)
- Gerald Pfeifer (SUSE)
- Joel Sherrill (OAR Corporation)
- Jim Wilson (Specifix Inc)

O que é hoje

Documentação http://gcc.gnu.org/onlinedocs/

Manuais do GCC 4.1.1 (pdf, ps ou html.tar.gz)

- GCC 4.1.1 Manual
- GCC 4.1.1 GFORTRAN Manual
- GCC 4.1.1 GCJ Manual
- GCC 4.1.1 CPP Manual
- GCC 4.1.1 GNAT Reference Manual
- GCC 4.1.1 GNAT User's Guide
- GCC 4.1.1 GNU JAR Manual
- Fontes texinfo de todos os manuais

Roteiro

GCC Development Mission Statement

- parte do Projeto GNU
- melhorar os compiladores usados no sistema GNU
- ambiente de desenvolvimento aberto com suporte a muitas plataformas
  - produzir um compilador otimizador de nível global
  - atrair grande número de desenvolvedores
  - garantir o funcionamento do sistema GNU em múltiplas arquiteturas e ambientes
  - testar e extender as características do GCC da forma mais ampla possível

GCC Development Mission Statement

Projeto de Software Livre

- suporte aos objetivos do projeto GNU como definidos pela FSF
- compiladores disponíveis nos termos da GPL
- *copyrights* da FSF
- outros componentes (bibliotecas, *testsuites*, etc) disponibilizados sob várias licenças livres, com *copyrights* dos autores ou da FSF
- relacionamentos legais com contribuidores são responsabilidade da FSF
- patches devem ser legalmente aceitáveis para serem incluídos no projeto GNU

GCC Development Mission Statement

Objetivos de Projeto e Desenvolvimento

- novas linguagens
- novas otimizações
- suporte para novas arquiteturas
- melhorias nas bibliotecas
- ciclo de depuração mais rápido
- diversas melhorias na infraestrutura

GCC Development Mission Statement

Ambiente de Desenvolvimento Aberto

- encorajar a cooperação e comunicação entre desenvolvedores
- trabalhar mais proximamente dos "consumidores"
- código aberto e abertura para novos desenvolvedores
- listas de e-mail abertas
- ferramentas e procedimentos amigáveis para os desenvolvedores
- última instância para resolução de conflitos é o Comitê Gestor

GCC e EGCS

GCC

- 1987: 1.0
- Richard Stallman (FSF)

EGCS ("eggs")

- Fork: 1997 (entre o GCC 2.7.2.3 e 2.8.0)
- *Re-emerged*: Abril 1999 (GCC 2.95)
- 1991: GCC 1.x estável mas com limitações na arquitetura
- meados dos anos 90: GCC 2.x fortemente controlado pela FSF
  - dificuldade para suporte outras linguagens
  - dificuldade em ter seu trabalho aceito
  - frustração
- 1997: EGCS *merge* de vários ramos experimentais
- EGCS demonstrou muito maior vitalidade
- 1999: EGCS torna-se GCC oficial

Roteiro

Estrutura

- interface externa padrão UNIX
- executa-se um *driver* que interpreta os argumentos, decide o compilador, executa o montador e eventualmente o *linker*
- para cada linguagem o compilador é um programa separado
- estrutura interna comum com um *front end* para cada linguagem que faz a análise sintática e gera uma AST
- um *back end* que converte a AST em RTL (*Register Transfer Language*), executa várias otimizações e produz um código assembly específico para cada arquitetura/SO
- é quase todo escrito em C, mas boa parte do *front end* de Ada é escrito em Ada

Front ends

- até recentemente a representação da árvore sintática não era totalmente independente da linguagem, de fato, uma parte do *back end* dependia da linguagem
- 2005: introdução de duas árvores independentes da linguagem
- GENERIC: o parser é agora responsável por criar a árvore sintática dependente da linguagem e convertê-la para este formato
- GIMPLE: gerada a partir da forma GENERIC, permite muitas otimizações globais independentes de linguagem e arquitetura
- este processo intermediário de otimização na árvore tem sido chamado de *middle end* e incluem:
  - propagação de constantes
  - eliminação de código morto
  - eliminação de redundância parcial
  - alocação de registradores

Back ends

- parcialmente especificados por macros do pré-processador e funções específicas para uma dada arquitetura: endianness, tamanho da palavra, convenção de chamada de funções
- a parte inicial do *back end*, que gera RTL, usa estas informações, portanto RTL não é tão independente do processador como deveria
- otimizações na RTL, como otimização de desvios, eliminação de sub-expressões comuns, perderam parte de sua importância devido às otimizações globais baseadas em SSA (*static single assignement*) da forma GIMPLE
- uma fase de "recarga"altera os registradores abstratos para registradores reais, com base nos padrões de descrição dos conjuntos de intruções das máquinas alvo
- a fase final é relativamente simples, pois as escolhas mais importantes já foram feitas na fase de "recarga", e reduz-se à montagem dos strings que representam as instruções

Roteiro

Cygwin = GNU + Cygnus + Windows

O que é Cygwin?

- Cygwin é um ambiente *a la* Linux para Windows
  - DLL, cygwin1.dll, que atua como uma camada de emulação Linux
  - uma coleção de ferramentas Linux

O que Cygwin não é

- não é uma forma de rodar aplicações nativas Linux em Windows, elas têm que ser recompiladas
- não é uma maneira de fazer aplicações nativas Windows aptas a usar funcionalidades do Linux

MinGW - Minimalist GNU for Windows

- GCC nativa em Windows
- coleção de arquivos de cabeçalhos específicos para Windows que podem ser disponibilizados e distribuídos gratuitamente
- GNU *toolsets* para produzir aplicações nativas Windows
- não depende de DLLs de tempo de execução para rodar (contraposição direta ao modelo do Cygwin)

MSYS - Minimal SYStem

- prover a abilidade de executar e criar `Makefiles` por scripts `POSIX/Bourne configure`
- uso de `bash` no Windows

Roteiro

Otimização no C/C++

- -O
  Não otimiza, é o default.
- -O, O1
  Otimiza. Vale lembrar que a otimização torna a compilação mais lenta e consome muito mais memória, principalmente para funções grandes.
  Com -O,-O1, o compilador tenta minimizar tamanho e tempo de execução, sem sobrecarregar o tempo de compilação.
  - -fdefer-pop: deixa acumular argumentos de várias chamadas de função na pilha e desempilha em grupo;
  - -fdelayed-branch: se suportado na arquitetura tenta reordenar instruções para aproveitar slots disponíveis após instruções de desvio retardadas

Otimização no C/C++

- -fguess-branch-probability: tenta identificar, usando informações disponíveis qual o resultado mais provável de um desvio para evitar falha de pipeline
- -fcprop-registers: desabilita passo de propagação de cópia após alocação de registradores
- -floop-optimize: move expressões constantes para fora de laços, simplifica testes de condição de saída e opcionalmente executa
- -fif-conversion: tenta converter desvios condicionais em equivalentes sem desvio, depende de -fif-conversion2
- -fif-conversion2: usa execução condicional, onde disponível para transformar desvios condicionais em equivalentes sem desvio
- -ftree-ccp: propagação de constantes condicionais esparsas
- -ftree-dce: eliminação de código morto

Otimização no C/C++

- -ftree-dominator-opts: variedade de limpeza escalar simples (propagação de contantes/cópia, eliminação de redundâncias, propagação de intervalos e simplificação de exoressões
- -ftree-dse: eliminação de área de armazenamento morta
- -ftree-ter: troca de expressões temporárias durante a fase SSA− >normal
- -ftree-lrs: tempos de vida diferentes de uma variável são desmembradas em variáveis únicas, para facilitar otimização posterior
- -ftree-sra: troca de agregados escalares
- -ftree-copyrename: tenta renomear temporários do compilador para outros nomes de variáveis, mais próximos dos nomes originais

Otimização no C/C++

- -ftree-fre: eliminação de redundância completa, considera expressões computadas em todos os caminhos da árvore
- -ftree-ch: cópia de cabeçalhos em árvores, aumenta e efetividade de otimização de movimentação de código, bem como economiza um desvio
- -fmerge-constants: tenta eliminar constantes (ponto flutuante e strings) duplicadas entre unidades de compilação