Universidade Federal de Minas Gerais - UFMG

Departamento de Ciência da Computação - ICEX

Começando a desenvolver no compilador LLVM e iniciando a implementação de um alocador de registrador

Matheus Lima Diniz Araújo

Índice:

Introdução

Neste documento será apresentado de forma simples e objetiva dos primeiros passos no desenvolvimento de um novo Alocador de Registradores para o compilador LLVM.

Para tornar mais claro este guia, há uma introdução sobre o que é o compilador LLVM e porque escolher este compilador para desenvolver não só alocadores de registradores mas qualquer outra otimização para compiladores. Discutiremos também os atuais alocadores de registradores que compõe as versões padrões do LLVM e por fim será apresentado um passo a passo de como começar a implementação de um novo alocador de registradores para o LLVM.

LLVM – Low Level Virtual Machine:

O LLVM é uma infraestrutura de compilador escrita em C++ que foi desenvolvida com o propósito de otimizar em tempo de compilação, ligação e execução programas escritos em diferentes linguagens, como C, C++,Pyton, dentre outras.

Atualmente o LLVM está sobre os domínios da Apple Inc.

A estrutura do LLVM:

O LLVM pode ser dividido por 2 partes básicas: FrontEnd e BackEnd, esta forma de divisão é comum para praticamente todos os compiladores.

FrontEnd:

Analisa o código fonte e gera a representação interna(ou representação intermediária) do programa para o compilador(IR). A representação interna do LLVM é baseada no padrão SSA, Static Single Assigment, está representação está comum para todas as partes da compilação do programa.

BackEnd:

O BackEnd do LLVM analisa o IR gerado no FrontEnd, otimiza-o e depois realiza a geração de código (code generation), este último último gera o código de maquina final para um determinado sistema.

Este guia focará na parte da otimização do BackEnd, para ser mais especifico na parte de otimização da alocação de variáveis do programa a ser compilad nos registradores do sistema.

Passes:

Os passes são pequenos módulos de código que juntos formam praticamente todo o LLVM. Para realizar qualquer modificação, seja para otimizar ou desenvolver novos recursos ao LLVM, é preciso saber como implementar estes passes. Alocadores de Registradores, Geradores de Código de Máquina, ou mesmo um contador de funções para um programa pode ser um pass.

Com este sistema de passes, o LLVM possui a característica de ser muito modular, demonstrando um código intuitivo, e assim tornar relativamente fácil realizar otimizações e modificações.

Porque o LLVM?:

- O LLVM é um compilador em constante de desenvolvimento portanto recebendo otimizações constantemente.
- Estrutura bastante modular, com um sistema de passes eficiente.
- Código bem escrito e comentado.
- Possui uma comunidade bem entusiasmada onde são trocadas informações, perguntas e respostas. Na lista: http://lists.cs.uiuc.edu/mailman/listinfo/llvmdev é possível encontrar diversas soluções para problemas relacionados ao desenvolvimento de recursos para LLVM.

Alocadores de Registradores

A alocação de registradores é a fase do compilador que decide quais valores do programa devem ser atribuídos aos registradores físicos do processador. Geralmente, existem menos registradores na máquina do que o necessário para guardar todas as variáveis de um programa em um determinado instante da execução, fazendo com que alguns valores contidos em registradores sejam derramados para a memória. Quanto mais instruções de acesso á memória existirem no código, pior será a eficiência do programa.

Uma alocação de registradores eficaz reduz o número deste tipo de instruções. Porém este processo é NP-Completo, portanto é necessário criar heurísticas para aproximar da solução ótima.

O processo de alocação pode ser reduzido intuitivamente a um processo de coloração de grafos, onde o número de cores seria o número de registradores e os vértices seriam as variáveis. Estes vértices possuiriam relações quando as variáveis correspondentes estivessem vivas ao mesmo tempo.

A maioria dos alocadores de registradores utilizam esta técnica para resolver o problema e realizar o menor número de derramamentos possível para a memória. Apesar da técnica de coloração de grafos ser intuitiva, o tempo de alocação para algoritmos baseado neste método é relativamente caro. Como exemplo temos o Linear Scan que utiliza outra heurística para o problema.

Alocadores de Registradores padrões nas versões do LLVM:

- **Linear Scan Register Allocator:**

  O Linear Scan usa um algorítimo de alocação de registradores globais, não baseado na coloração de grafos. Ele se comporta da seguinte maneira:

  Dado o live range(tempo de vida) das variáveis em uma função, o algoritmo do Linear Scan escaneia todos os lives ranges alocando as variáveis de forma gulosa. O algoritmo é simples, eficiente e produz um código relativamente limpo. É útil em situações que requerem um bom tempo de compilação e um claro entendimento.

  O Linear Scan era usado como o Alocador de Registrador *default* do LLVM até a versão 2.8.

- **Greedy Register Allocator:**

  O Greedy Register Allocator como o próprio nome já menciona, também é baseado em soluções gulosas para o problema da alocação de registradores. Ele possui um comportamento semelhante ao Linear Scan mas é um pouco mais esperto. Ele possui as seguintes principais características:

  - Gera binários em média 2% menores e 10% mais rápidos.
  - É mais rápido que o Linear Scan em tempo de compilação para programas que possuem funções pequenas, mas perde ao compilar grandes funções.

  A partir da versão 2.9 do LLVM,o Greedy Register Allocator, se tornou o Alocador de Registradores *default*.

- **Basic Register Allocator:**

  O Basic Register Allocator é um alocador básico para o LLVM, de forma a ser considerado um exemplo didático de como criar um alocador de registradores para o mesmo.

  O Basic não promove uma otimização de código e é inviável para o uso na compilação de programas. Ele analisa a vida dos registradores e derrama-os (spills) sempre que não há registradores disponíveis, se mostrando uma péssima heurística.

  O seu código bem comentado é provido no Link: http://llvm.org/docs/doxygen/html/RegAllocBasic_8cpp_source.html

  Utilize-o como base para implementar seus próprios alocadores.

Instalando e configurando o LLVM

O LLVM é um compilador complexo, portanto possui inumeráveis configuração de compilação, instalação e execução. Esta é mais uma característica que comprova a sua "maleabilidade" para diversas arquiteturas e linguagens.

Nesta seção focarei em apresentar uma forma básica para compilar e executar o LLVM a partir do seu *source*.

1. Primeiramente é necessário fazer o download do *source* code, no site http://llvm.org/releases/ você encontrará as últimas versões do LLVM, selecione a versão mais nova. Deixo claro que este tutorial foi realizado baseado na versão 2.9. Após o download, descompacte os arquivos(a pasta criada será o *root* do LLVM) e via terminal, realize os comandos:

   ./configure
   make

   O primeiro comando tem o objetivo de configurar o source code do LLVM a ser compilado, de modo a adequar ao seu sistema. O segundo comando compila o código.

2. Fazendo estes passos você já possui o BackEnd do LLVM instalado e utilizável em sua máquina. Mas não adianta tentar compilar um código em C por exemplo, pois é necessário o FrontEnd do compilador.

   Para tal, é necessário fazer o download do source code do LLVM-GCC FrontEnd na mesma página do download anterior.

   Para compilar e executar, basta seguir as instruções do arquivo README.llvm contido no download acima. Siga estritamente os comandos neste arquivo,principalmente na hora de escolher a arquitetura do sistema, pois caso contrário o LLVM-GCC pode não compilar.

Pronto. Se você seguiu corretamente os passos, vá ao diretório install (que você criou) e via terminal acesse o diretório bin. Tente compilar um arquivo .c com o binário *llvm-gcc,* usando os mesmos argumentos do gcc, para assim certificar que tudo foi instalado corretamente.

Obs: Utilize o llvm-gcc para gerar bitecodes(.bc) dos programas a serem compilados

Implementando o seu primeiro PASS

Nesta seção discutiremos a estrutura básica de um *Pass* por meio de um exemplo bem simples, depois será discutida como deveria ser a estrutura de um pass para alocadores de registradores.

Crie um diretório dentro do diretório em algum lugar no root do LLVM para que sejam, guardados os passes criados.

Makefile:

Para que o seu pass seja compilado e linkado à estrutura do LLVM é necessário criar um Makefile para ele com o seguinte conteúdo:

```
# Makefile for hello pass

# Caminho até o diretório de maior nível do LLVM
LEVEL = ../../..

# Nome da Biblioteca a ser construída
LIBRARYNAME = Hello

# Fazer com que o pass seja carregado por ferramentas do LLVM
LOADABLE_MODULE = 1

# Incluir o makefile na implementação
include $(LEVEL)/Makefile.common
```

Pass Básico:

O pass que será explicado possui a simples função de imprimir no terminal o nome das funções(não externas) que o programa a ser compilado possui. Desta forma o pass não modifica o programa, apenas o inspeciona. Vamos chamar o arquivo que conterá o pass de *hello.cpp.*

```
1. #include "llvm/Pass.h"
2. #include "llvm/Function.h"
3. #include "llvm/Support/raw_ostream.h"
4. using namespace llvm;
5. namespace {
6.   struct Hello : public FunctionPass {
7.       static char ID;
8.       Hello() : FunctionPass(ID) {}
9.       virtual bool runOnFunction(Function &F) {
10.           errs() << "Hello: " << F.getName() << "\n";
11.           return false;
12.         }
13.       };
14.
15.       char Hello::ID = 0;
16.       static RegisterPass<Hello> X("hello", "Hello World Pass", false,
   false);
17.     }
```

Linhas de 1 a 3:

É necessário incluir os arquivos:

- Pass.h, pois estamos escrevendo um pass.
- Function.h, pois estamos operando sobre as funções do programa
- raw_ostream.h,pois estamos imprimindo algo.

Linha 5:

É iniciado um namespace anonimo. Namespace anonimo em C++ é equivalente à keyword

“static” em C(no escopo global). Desta forma torna as coisas que são declaradas dentro deste namespace apenas visível pelo arquivo corrente.

Linha 6: Declaração do pass em si. Declara que a classe “Hello” que estamos criando é subclasse da classe *FunctionPass*. Classe esta responsável por operar funções do programa compilado.

Linhas 7 e 8: Declara um identificador para o pass. Cada pass do LLVM possui seu identificador. O ID é configurado em FunctionPass().

Linhas 9 e 13: Sobrescreve o método abstrato r*unOnFunction(),* herdado da classe FunctionPass. É dentro deste método que é implementado toda a função do pass a ser gerado. No exemplo é apenas impresso o nome das funções do programa compilado(F.getName()) na saída *errs()*.

Linha 15: O ID do nosso pass é inicializado. O LLVM usa o endereço do ID para identificar o pass e não o seu conteúdo, portanto podia ser qualquer valor diferente de 0.

Linha 16: Por último registramos a nossa classe *Hello,* dando a ela um argumento que será usado na linha de comando ao compilar o *pass*(no caso *hello*), o nome de nosso *pass* será “*Hello World Pass”*, os últimos 2 argumentos descreve o seu comportamento. O primeiro **false** representa que o pass irá não somente passar pelo CFG(Control Flow Graph), então é false. O segundo **false** significa que o nosso pass não é um AnalysisPass(somos *FunctionPass)*.

Compilando e executando o PASS no LLVM:

Agora que você possui o seu pass hello.cpp e o Makefile, basta realizar o comando:

```
./gmake
```

Assim será gerado o arquivo *Hello.so* no diretório ($LEVEL)/Debug+Asserts/lib/, este arquivo é o seu pass compilado.

Para executá-lo use os seguintes comandos:

```
$ opt -load ../../../Debug+Asserts/lib/Hello.so -hello < hello.bc > /dev/null
```

Este comando usa a ferramenta do LLVM *opt* para acessar o seu pass, da seguinte forma: -*load argumento*, onde *argumento* é o caminho de seu pass.

-*hello*, quando o nosso pass foi registrado, este comando passa a ser válido e assim o pass *hello* é executado sobre o bitecode *hello.bc*. Este *hello.bc* é um programa que foi gerado a partir de uma compilação do FrontEnd (o LLVM-GCC gera arquivo .bc a partir de arquivos .c).

/dev/null significa ignorar os resultados do opt.

Após executar este comando, será impresso no terminal os nomes das funções que *hello.bc* possui. Concluindo assim o objetivo do pass.

Pass para um Alocador de Registrador:

Nesta seção será somente discutido uma rápida introdução a respeito de como os Alocadores de Registradores funcionam no LLVM. Somente com este material não será possível implementar um alocador de registrador, mas será mostrado alguns pontos necessário no desenvolvimento do mesmo.

Vamos supor que o nosso alocador de registrador chame "Exemple Register Allocator".

Configurando o LLVM para reconhecer o seu pass:

É necessário saber que no arquivo ../lib//include/llvm/CodeGen/passes.h existe a declaração de todos os Passes que serão utilizados pelo BackEnd do LLVM, por tanto é necessário que a declaração da função que cria o seu pass esteja lá. Por exemplo:

```
Functivos *createExampleRegisterAllocator();
```

Agora o PASS do alocador "existe" para o LLVM, mas ainda é necessário configurar para que ele seja utilizado na alocação de registradores. A função que determina qual alocador será utilizado de acordo com a preferência do usuário é a *createRegisterAllocator()* que está implementada no arquivo ../lib/CodeGen/*passes.cpp,* se deseja criar um novo alocador de registrador é necessário configurar este arquivo, incrementando-o com o seu alocador.

Estruturas do PASS que possui um alocador de registradores:

Entrada:

Para entender como um alocador de registrador funciona na prática, é preciso saber como é realizado as entradas do seu algoritmo. Em um compilador, os alocadores utilizam como entrada o que chamamos de Basic Blocks, os quais são blocos de instruções que normalmente correspondem a uma função ou rotina em uma determinada linguagem. A partir da análise desses Basic Blocks, o alocador pode observar a duração dos live range das variáveis e assim saber quantas e quais variáveis estão vivas em um determinado instante da execução do programa.

Vale destacar que o LLVM possui uma biblioteca muito útil para realizar análises de tais Basic Block, no arquivo ../lib/CodeGen/*RegAllocBase.h* pode-se encontrar funções extremamente úteis, as quais economizam muito tempo na implementação das estruturas do seu pass. Em ../lib/CodeGen/MachineBasicBlock.cpp está disponível um grande leque de funções para manipular os Basic Blocks.

Heurística:

Com o conhecimento de como é realizado a entrada, parti-se para a implementação do que realmente compõe a heurística do alocador de registradores a ser implementada. Possuindo a heurística em mente, basta convertê-la para a estrutura do LLVM.

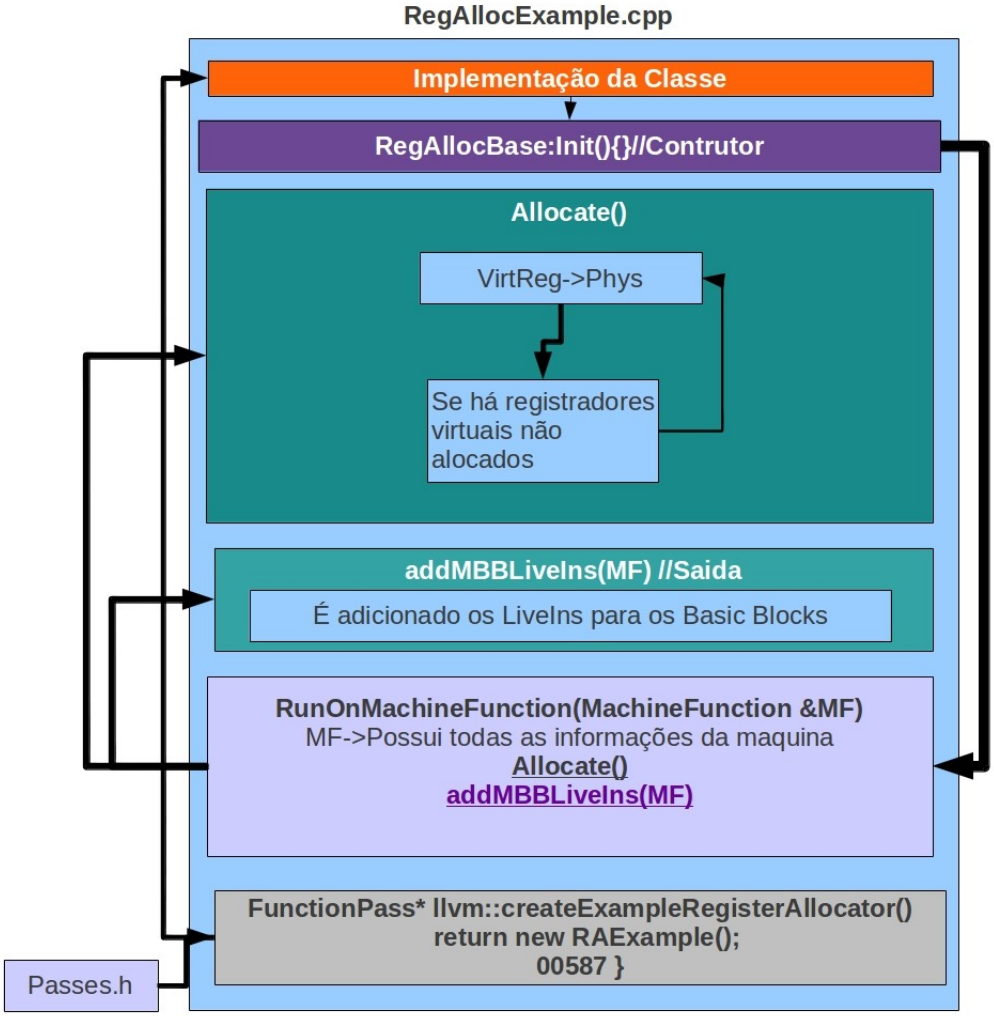
Para ajudar neste empreendimento o uso do Basic Register Allocator (../lib/CodeGen/RegAllocBasic.cpp) é fundamental e extremamente recomendado, também é importante observar o comportamento dos outro alocadores como o Linear Scan (../lib/CodeGen/RegAllocLinearScan.cpp).

Baseando nos atuais alocadores é possível se ter uma noção de como o PASS deve se comportar e como conseguir determinadas informações do LLVM que podem ser úteis para a sua heurística, como o sistema de *Coalescing , PhiElimitation e* Spills, presentes na maioria dos alocadores descritos.

Saída:

Como saída do alocador, tem-se as variáveis derramadas para a memória e determinadas variáveis associadas aos registradores. Uma função virtual importante que está contida no arquivo RegAllocBase.h é a *selectOrSpills(),* ela seleciona quais variáveis devem ser derramadas e quais devem receber os alocadores. O RegAllocBase.h possui a função addMBBLiveIns(), cuja finalidade é associar cada vida de uma variável escolhida a um registrador físico. Em relação aos derramamentos para a memória temos a função spillInterferences() ou Spiller().

Na imagem abaixo pode-se observar graficamente como seria composto o nosso pass. A classe do alocador seria criada pela função *FunctionPass** que você declarou lá no *passes.h* explicado anteriormente. Ao ser criada a classe, o construtor é ativado, a maioria dos alocadores utilizam a função Init() do *RegAllocBase* para implementar o seu construtor. Quando necessário, o compilador invocará o seu compilador, e executará a função runOnMachineFunction(), está função é o esqueleto do seu alocador, ela executa a entrada, o algoritmo da heurística escolhido e por fim associa os LiveRanges das variáveis aos registradores.



Recomendações de estudos ao implementar um Alocador:

Para implementar um alocador de registradores é muito importante saber como o mesmo funciona na teoria, de forma a focar na heurística desejada. Mas também é necessário saber como ele funciona na prática dentro de um compilador. Um bom livro que trata de ambos assuntos é o **Modern Compiler Implementation in Java, Andrew W. Appel and Jens Palsberg** , nele é detalhado a implementação de um alocador de registrador de forma bastante clara e exemplificada.

É importante frisar que a lista de e-mail llvmdev@cs.uiuc.edu é bastante movimentada, nela diversos assuntos são discutidos e muitas duvidas podem ser sanadas.

O próprio site http://llvm.org/docs/ possui diversos artigos e tutoriais para que seus usuários possam aproveitar ao máximo a infraestrutura do LLVM. Para ser mais específico sugiro foco nos seguintes tópicos:

- Introduction to the LLVM Compiler System
- The LLVM Getting Started Guide
- Writing an LLVM Pass
- Writing an LLVM Backend
- The LLVM Target-Independent Code Generator

Por fim, e ao meu ver, o melhor material para estudos são os próprios códigos que compõem o LLVM, a sua clareza é indiscutível.

Conclusão:

Neste guia, foram apresentados conceitos elementares em relação a como é um compilador, em especifico o Low Level Virtual Machine, LLVM. Foram discutidas as estruturas que um o LLVM possui e as características que motivam a escolhê-lo como base para implementar códigos para otimizações em compiladores.

Procurei discutir como seria realizado a implementação de um alocador de registradores no LLVM. Inicialmente é preciso ter em mente como é o desenvolvimento de um PASS, para que então possamos focar nos conceitos e diretrizes necessárias na criação de um alocador de registradores.

Deixo claro que este guia está longe de ser um manual para o desenvolvimento de alocadores de registradores para o LLVM, mas a partir dele podemos ter uma real noção dos primeiros passos necessários para se tornar um bom desenvolvedor de uma ferramenta extremamente poderosa que é o LLVM.