Pontifícia Universidade Católica de Minas Gerais
Departamento de Ciência da Computação
Disciplina: Compiladores - Trabalho Prático 1 - Grafo de Fluxo de Controle
Professor: Pedro Ramos

VALOR: 10 PONTOS
ENTREGA: 06/05/2024

Grafo de Fluxo de Controle (CFG)

Otimizadores e seres humanos não enxergam um programa de computador da mesma maneira. Seres humanos (nós) estão mais interessados no código fonte. Porém, código fonte é extremamente diferente de código de máquina. Além disso, sob um ponto de vista de engenharia, é melhor que exista uma maneira comum de representar programas que seja comum à várias linguagens e arquiteturas. Nesse sentido, o otimizador precisa trabalhar com uma representação intermediária do programa.

Usualmente, compiladores representam programas como Grafos de Fluxo de Controle.

Um CFG é um grafo direto em que os nós são blocos básicos. Existe uma aresta que vai do bloco básico B1 ao bloco básico B2 se a execução do programa pode fluir de B1 até B2.

Veja por exemplo o código abaixo, que calcula o fatorial de um número, e seu respectivo CFG.

```
1 int fact(int n) {
2     int ans = 1;
3     while (n > 1) {
4         ans *= n;
5         n--;
6     }
7     return ans;
8 }
```

Nos dois trabalhos práticos desta disciplina, trabalharemos com um arcabouço de compilação chamado LLVM (Low Level Virtual Machine). Apesar do nome, o LLVm é bem mais que uma máquina virtual de baixo nível: é um compilador e um otimizador de programas C e C++. Tem um ecosistema vivo e uma comunidade ativa de engenheiros de compiladores que trabalham duro para criar novos passes e otimizações para tornar o código objeto mais eficiente. LLVM hoje é o arcabouço mais utilizado na área de compiladores, e é utilizado também como compilador oficial em muitas empresas importantes: Apple, Google, Meta.

Na Figura 2 vemos o pipeline do LLVM. O front-end (clang) transforma código C em bytecode (representação intermediária). O otimizador (opt) trabalha em cima dessa representação intermediária, realizando diversos passes, ou otimizações, como Propagação de Constantes, Eliminação de Código Morto, entre outras. Note que o otimizador executa várias otimizações em uma ordem arbitrária, pois é impossível determinar a sequência ótima de otimizações. Note também que, por causa disso, a saída do otimizador está sempre no mesmo formato da entrada, pois o código pode passar por ele infinitas vezes. O opt possui 3 níveis de agressividade, sendo o nível O3 aquele com passes e transformações mais agressivas, que podem modificar a semântica do seu programa drasticamente.

Infelizmente, não é tão trivial identificar quais passes são executados nos níveis O1, O2 e O3 do

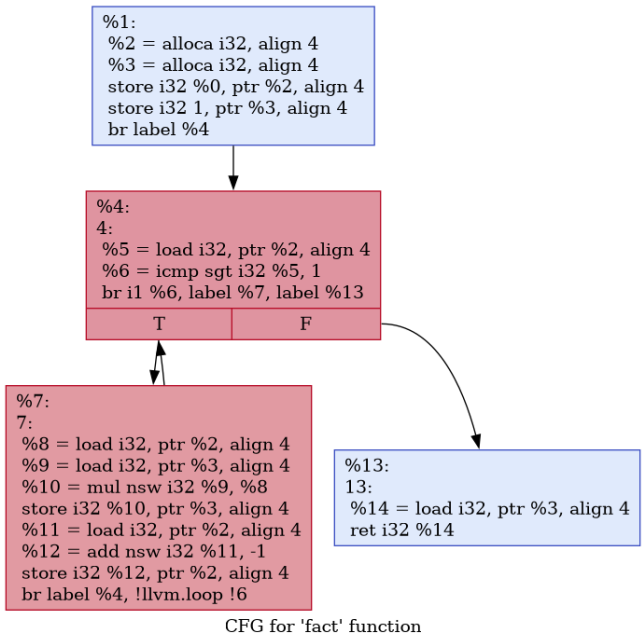

Figura 1: Grafo de fluxo de controle para o programa acima.

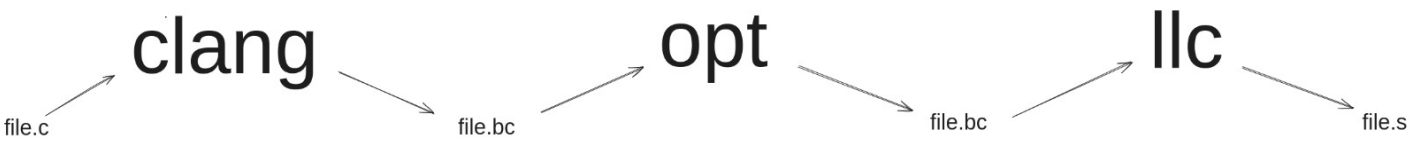

Figura 2: Pipeline do LLVM.

clang. Neste arquivo é possível notar que várias trasnformações são adicionadas ao pipeline dependendo da variável `OptLevel`.

Por fim, após aplicadas as otimizações, o compilador estático (llc) transforma o bytecode em código de máquina para diversas arquiteturas, além de realizar otimizações dependentes de máquina (Alocação de Registradores, por exemplo).

Requisitos do trabalho.

Você deverá escrever um script em `bash` ou `python` que recebe como entrada um código fonte C (`fonte.c`) e produz, na saída, imagens no formato PNG (`main.png, foo.png, ...`) que representam os Grafos de Fluxo de Controle para as funções contidas no arquivo fonte.

No arquivo `cfg.zip` você vai encontrar uma pasta `src` contendo alguns programas. Você deve gerar os CFG's para cada um dos programas contidos na pasta.

Além disso, você deve gerar também versões diferentes do CFG para cada nível de otimização do clang. Portanto, para cada código fonte, você deve gerar 4 versões de CFG: um CFG para o nível de otimização O1, outro CFG para o nível O2, e outro CFG para o nível O3. Lembrando que pode haver mais de um CFG por arquivo, uma vez que os grafos são por função e não por módulo.

Por fim, você deve observar os grafos gerados e explicar pelo menos 2 transformações ocorridas em níveis de otimização diferentes.

O que deve ser entregue:
Você deve entregar um arquivo .zip contendo:

1. Um script `gen_cfg.sh` ou `gen_cfg.py` que recebe como primeiro arqumento um programa escrito em C e gera, como saída, imagens PNG representando o grafo de fluxo de controle deste programa em três níveis de otimização diferentes (O1, O2 e O3). [6 pts].

2. As imagens dos CFG's para todos os programas contidos na pasta `src/`, nos três níveis de otimização diferentes. [1 pt].

3. Um PDF descrevendo **duas ou mais** transformações que são possíveis notar ao analisar o CFG de um mesmo programa após níveis de otimização diferentes. [3 pts].

Considerações finais && Dicas

1. Uma vez que você tenha o clang instalado, você pode gerar um arquivo da representação intermediária legível com a flag `-emit-llvm`.

2. Existe um passe que compila a representação intermediária para o formato `.dot`. Você consegue achar esse passe? Tente usar o comando `opt --help` para listar todos os passes possíveis.

3. A ferramenta graphviz é o software ideal para lidar com arquivos `.dot`.

4. Veja os exemplos que já tem CFG's formados `infinity.c & fact.c`.

5. **Atenção!** Se a representação intermediária tiver a flag `optnone` a ferramenta opt irá ignorar quaisquer tentativas de rodar algum passe, incluindo os de geração de arquivos .dot.

6. Caso algum dos programa não compile, explique no pdf submetido qual problema houve na compilação do mesmo.

7. **Atenção!** Há mais de um arquivo .c na pasta `singlesource/`

Bom trabalho!