O Impacto das Otimizações Aplicadas Por um Compilador Otimizante no Contexto de um Processador Embarcado

Leonardo Deganello de Souza (PIC /Uem), Anderson Faustino da Silva (Orientador), e-mail: anderson@uem.br

Universidade Estadual de Maringá / Centro de Tecnológia/Maringá, PR

Área e subárea do conhecimento: Ciências Exatas e da Terra / Ciência da computação

Palavras-chave: Compiladores, Otimização, Processador ARM

Resumo:

Códigos gerados por compiladores podem não ter a melhor qualidade possível, devido à dificuldade de obter a sequência de instruções ótima em meio a inúmeras possibilidades. Desenvolvedores de compiladores procuraram melhorar a qualidade do código gerado mediante a implementação de inúmeras otimizações. A aplicação de otimizações, porém, pode prejudicar a qualidade do código, se mal utilizada. Portanto, se faz necessário analisar o impacto de cada otimização provida pelo compilador e desta forma montar o melhor conjunto possível de otimizações. Este trabalho analisa o impacto de diferentes otimizações de códigos no contexto do processador ARM.

Introdução

Compilador é uma ferramenta que geralmente traduz código de linguagens de alto nível (SCOTT, 2009), legíveis a seres humanos, em linguagem de montagem (Assembly) (BLUM, 2005), a qual pode ser facilmente transformada em código binário por um utilitário montador (Assembler). Essa tradução não é trivial e é feita em etapas, que podem variar de acordo com a implementação do compilador (AHO, 2004). No entanto, independente da quantidade de etapas, estas são divididas em fases de análise e fases de tradução, as quais se comportam como segue:

- Fases de análise realizam verificações a procura de possíveis erros no código fonte. Em geral, os compiladores implementam as seguintes análises: léxica, sintática e semântica.
- Fases de tradução realizam transformações no código fonte até que este se torne em código em linguagem de montagem. Nessa fase é possível implementar as seguintes etapas: geração de uma representação intermediária, manipulações de blocos básicos, aplicação de otimizações, seleção de instruções para a máquina alvo,

alocação de registradores e emissão de código em linguagem de montagem.

O que ocorre no processo de compilação, em geral, é que um pequeno trecho de código fonte é transformado em uma sequência maior de instruções escritas em linguagem de montagem. Assim, para um determinado código fonte é possível gerar várias sequências distintas, as quais produzem o mesmo resultado semântico. A escolha de tal sequência implica diretamente nos seguintes fatores: tamanho do código alvo, velocidade de execução, consumo de energia, consumo de memória, dentre outros fatores.

Compiladores que aplicam otimizações, denominados compiladores otimizantes, fornecem geralmente dezenas de otimizações. Dentre estas é possível escolher quais e em que ordem serão aplicadas (ALMAGOR e outros, 2004), pois nem toda otimização modificará o código de maneira benéfica, podendo em alguns casos até ocasionar perda de desempenho. Além disto, aplicar todas as otimizações possíveis não significa necessariamente obter o melhor código, dado que as características do código fonte podem não se adequar às características específicas de cada otimização.

Portanto, é essencial avaliar o impacto individual de cada otimização, em diferentes programas. Principalmente, no contexto de um processador embarcado pelo fato deste possuir recursos limitados, quando comparado com um processador utilizado por sistemas desktop.

Materiais e métodos

Para analisar o impacto de diferentes otimizações em diferentes programas, será utilizado o seguinte ambiente de execução:

- Compilador: LLVM 3.7 (LLVM, 2018)
- Otimizações: 130
- Processador: ARM Cortex-A7 quad-core 900 MHz
- Programas: Blowfishe, pgpe, rijndaele e sunsane.

O objetivo é analisar a redução no tempo de execução ocasionado por cada otimização individualmente, frente ao tempo de execução sem utilizar otimizações. Além, disto é importante analisar o tempo de execução quando é aplicado ao programa um conjunto de otimizações.

Resultados e Discussão

Figura 1 apresenta o padrão de aceleração obtido por cada otimização individualmente.

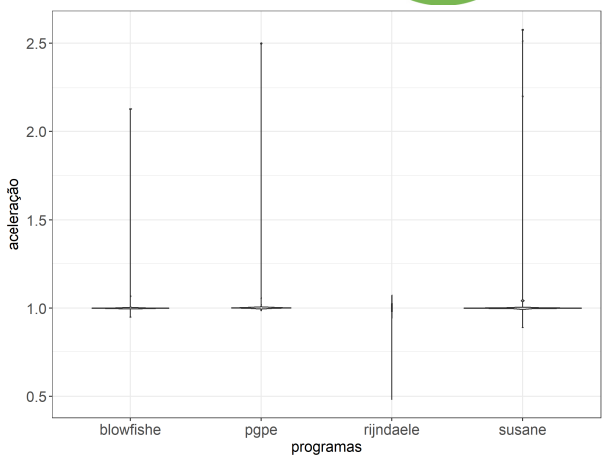

Figura 1 – Aceleração obtida por cada otimização avaliada.

É possível organizar as otimizações em três grupos distintos:

1. As que conseguiram uma aceleração igual ou ligeiramente superior a 1;
2. As que as otimizações pioraram a aceleração dos programas; e
3. As que obtiveram uma aceleração superior a 2.

O Grupo 1 concentra a maior parte das otimizações. Isto indica que tais otimizações não possuem impacto quando utilizadas individualmente. Portanto, tais otimizações devem ser utilizadas em conjunto com outras. O Grupo 2 concentra poucas otimizações (10), as quais ocasionam perda de desempenho quando utilizadas individualmente. Isto ocorre pelo fato de algumas otimizações acarretarem um efeito colateral quando aplicadas a um determinado programa. Tal efeito é geralmente o mal uso da cache, o que ocasiona um aumento na quantidade de falhas na cache e consequentemente uma degradação no desempenho.

O Grupo 3 é composto pelas otimizações mem2reg, scalarrepl-ssa, scalarrepl e sroa. Estas otimizações são eficientes para serem utilizadas individualmente, sendo capazes de alcançar uma aceleração superior a 2 frente a um programa compilado sem o uso de otimizações.

Outro ponto a ser destacado é o caso do programa rijndaele, para o qual nenhuma otimização obteve um impacto positivo, mesmo as pertencentes ao Grupo 3. Isto demonstra que a escolha da otimização que irá melhorar a qualidade do código final é um problema depende de programa.

A Figura 2 apresenta a aceleração obtida pelos conjuntos de otimizações O1, O2 e O3, frente ao tempo de execução sem utilizar otimizações.

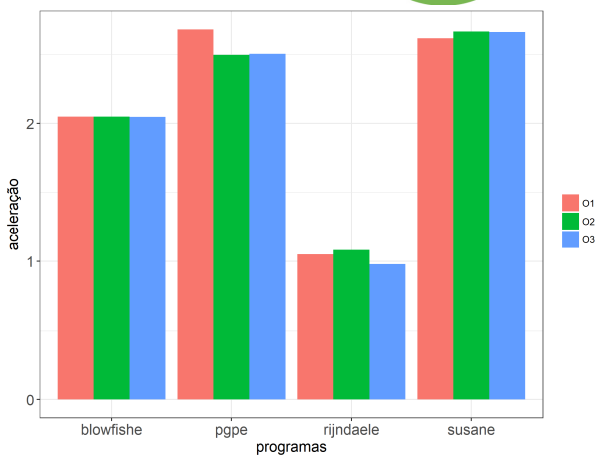

Figura 2 – Aceleração obtida pelos conjuntos de otimizações O1, O2 e O3.

Os resultados apresentados na Figura 2 indicam três pontos. Primeiro, é necessário utilizar um conjunto de otimizações para obter um código final de boa qualidade. Segundo, algumas otimizações individuais ocasionam um impacto semelhante a conjuntos de otimizações. Terceiro, o impacto ocasionado por um determinado conjunto de otimizações não será necessariamente o mesmo quando utilizado para compilar um diferente programa.

Conclusões

Este trabalho apresentou o desempenho obtido por diversas otimizações, no contexto do processador ARM A7. Os resultados indicam que é possível obter uma aceleração igual ou superior a 2, e que as otimizações mem2reg, scalarrepl-ssa, scalarrepl e sroa são eficientes para reduzir o tempo de execução de um programa. Além disto, este trabalho demonstrou que o problema de encontrar uma otimização efetiva (ou um conjunto) é um problema dependente de programa.

Referências

ALMAGOR, L., COOPER, K. D., GROSUL, A., HARVEY, T. J., REEVES, S. W., SUBRAMANIAN, D., TORCZON, L., WATERMAN, T. Finding effective compilation sequences. **ACM SIGPLAN Notices**, v. 39, n. 7, p. 231-239, 2004.

AHO, A. V., LAM, M. S., SETHI, R., ULLMAN, J. D. **Compilers:** Principles, Techniques, and Tools, 2 ed. Prentice Hall, Boston: USA, 2006.

BLUM, R. **Professional Assembly Language. Programmer to Programmer**. Wiley, Indianapolis, Indiana: USA, 2005.

LLVM. **Low Level Virtual Machine**. Disponível em **www.llvm.org**. Acesso em: 30 de julho de 2018.

SCOTT, M. L. **Programming Language Pragmatics**, 3 ed. Morgan Kaufmann Publishers Inc., San Francisco: USA, 2009.