Algoritmo VND Aplicado na Seleção de Otimizações do Compilador Clang

William Rodrigues da Silva, Graduando - Informática, UEM (Universidade Estadual de Maringá)

***Resumo*—O compilador clang nos permite selecionar um conjuto de otimizações para compilar um determinado programa, entretanto, dado um programa específico, nem todas as otimizações efetivamente melhoram o desempenho do mesmo. Este trabalho tem como objetivo desenvolver um mecanimso para obtenção de um bom conjunto de otimizações, que aplicado na compilação de um determinado programa, melhorem seu tempo de execução. Como resultado foram apresentados tempos de execução dos programas em média 24% mais rápidos quando comparados com programas compilados no conjunto de otimizações O3.**

***Index Terms*—Compilador, clang, seleção de otimizações, VND.**

I. Introdução

O compilador clang possui uma lista de otimizações que podem ser aplicadas na compilação de programas. Desta forma, ao compilar um programa com clang podemos selecionar um conjunto de otimizações a serem aplicadas no momento da compilação deste. Contudo, nem todas as otimizações disponíveis, efetivamente melhoram o desempenho daquele programa específico. Conforme é explorado nos resultados deste trabalho, algumas otimizações podem inclusive piorar o desempenho quando aplicadas em um programa. Neste caso, se faz necessário encontrar qual o melhor conjunto de otimizações, que efetivamente melhoram o desempenho quando aplicadas na compilação de um programa.

Este trabalho tem como objetivo desenvolver um mecanismo para obtenção de um bom conjunto de otimizações, que aplicado na compilação de um dado programa, melhorem seu tempo de execução. Para o desenvolvimento deste mecanismo vamos utilizar a meta-heurística VND (Variable Neighborhood Descent), tendo como universo de soluções o conjunto de otimizações O3 do compilador clang, e para auxílio na compilação e execução programas vamos utilizar o framework TF.

Como resultado foram apresentados os tempos de execução de 10 programas testados no algoritmo implementado, comparando o tempo de execução dos mesmos quando executados com compilação no conjunto O3 e com o conjunto gerado pelo alrogitmo, obtendo assim, em média, uma melhora de 24% no tempo de execução dos programas.

II. Desenvolvimento

Para realizar a seleção de um bom conjunto de otimizações na compilação de um programa utilizando o compilador clang, foi utilizado o algoritmo VND (Variable Neighborhood Descent) [1], tendo como universo de soluções todas as otimizações contidas no conjunto O3 do compilador.

Algorithm 1 VND: Compile Optimizations

$current\_level \leftarrow 1$
$best\_solution \leftarrow O3$
$current\_solution \leftarrow best\_solution$
while $curret\_level \leq max\_level$ **do**
  $neighborhood \leftarrow generate\_neighbor(current\_level)$
  $current\_solution \leftarrow local\_search(neighborhood)$
  if $time(current\_solution) \leq time(best\_solution)$ **then**
    $current\_level \leftarrow 1$
    $best\_solution \leftarrow current\_solution$
  else
    $current\_level \leftarrow current\_level + 1$
  end if
end while
return $best\_solution$

Figura 1. Pseudocódigo do algoritmo VND implementado.

Foram utilizados como estrutura de dados dois vetores de tamanho N, sendo N o número de otimizações disponíveis no conjunto O3 do clang. O primeiro vetor é preenchido com os nomes dos parâmetros de cada otimização, por exemplo -verify", e o segundo vetor é preenchido com valores booleanos. Desta forma, é estabelecida uma relação entre os dois vetores, mapeando se uma otimização está ou nao ativa por seus valores.

A Figura 1 mostra o pseudocódigo, com ideias gerais sobre o funcionamento do algoritmo VND implementado para solução do problema porposto.

O algoritmo inicia tendo como solução inicial O3, ou seja, todas as otimizações do conjunto O3 ativas, e a partir da solução inicial realiza uma busca local em sua vizinhança, buscando melhorar esta solução, que neste caso é comparada pelo tempo de execução do programa já compilado. Note que a geração da vizinhança de uma solução atual, é feita baseada no parâmetro de nível atual do algoritmo.

Para cada conjunto de otimizações gerados pelo algoritmo, é feita uma chamada ao framework TF para realizar a compilação do programa com aquele conjunto, e na sequencia o TF é chamado novamente, no modo de execução, realizando a coleta do tempo de execução do programa compilado com aquele conjunto. O modo de execução pode ser chamado várias vezes para então realizar a média do tempo de execução, isto pode ser parametrizado pelo argumento "accuracy", que

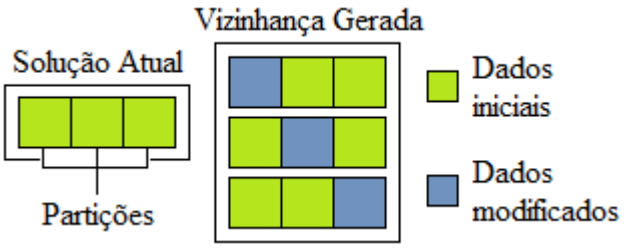

Figura 2. Mecanismo de partições na geração de vizinhanças

determina a precisão na média do tempo de execução coletado. Para os testes realizados neste trabalho foi utilizada uma precisão de 3, desta forma, a execução do programa é feita 3 vezes e então é retirada a média do tempo das 3 execuções.

A principal característica do algoritmo VND, consiste na manipulação do nível de vizinhança, isto é, vizinhanças de diferentes níveis são geradas durante sua execução.

O nível de vizinhança determina qual o tamanho da vizinhança gerada, desta forma, quanto maior o vível de vizinhança, mais elementos serão gerados no conjunto de vizinhos da solução atual.

A Figura 2 ilustra o mecanimsmo de partições para gerar uma vizinhança, ou seja, em cada nível do algoritmo, a solução atual é dividida em partições de tamanho N, e então são gerados vizinhos realizando modificações em cada partição. Assim, quanto maior for o nivel de vizinhança desejado, menor será o tamanho da partição N, gerando a partir da solução atual um número maior de vizinhos.

O algoritmo inicia com nivel de vizinhança 1, e sempre que encontra uma solução melhor, retorna ao nivel 1, entretanto quando a busca local não encotra nenhuma solução melhor entre seus vizinhos, o nivel de vizinhança é incrementado, permitindo que a busca local alcance vizinhos "mais distantes". Neste caso, o critério de parada fica relacionado ao nível de vizinhança, quando o nível máximo estipulado é atingido o algoritmo para, retornando a melhor solução registrada até o momento.

O nivel máximo de vizinhança gerada pode ser passado como parâmetro para o algoritmo e deve estar entre 1 e 5, neste trabalho utilizamos como padrão nivel 3, tendo em vista que quanto maior o nível de vizinhança, melhores poderão ser os resultados apresntados, porém o tempo para obte-los é maior.

III. Resultados e Conclusões

A seguir, são apresentados os resultados obtidos neste trabalho, para demonstração foram executados testes com o algoritmo desenvolvido sobre 10 diferentes programas, estes foram obtidos no próprio framework TF.

As estapas realizadas para apresentação dos resultados foram realizadas em um sistema operacional Ubuntu 18.04.1 de 64 bits no hardware que segue: 4,00 GB de memória principal e processador Intel Core i5-7400. Inicialmente o framework TF foi configurado para compilar e executar um único programa por vez, e então foram feitos os testes com cada programa, sendo eles: almabench, five11, CrystalMk, puzzle, dry, neural, Linpack, 01-qbsort, queens e Perlin.

Tabela I
Comparação de Desempenho (O3 x Conjunto gerado)

Programa	Tempo O3 (s)	Tempo Otimizado (s)	Melhora (%)
almabench	10.89	9.95	9.4
five11	2.55	2.26	12.8
CrystalMk	3.78	3.66	3.27
puzzle	0.059	0,033	78.7
dry	6.49	6.13	5.8
neural	0.148	0.138	7.2
Linpack	2.93	1.42	106.3
01-qbsort	0.147	0.143	2.8
queens	2.06	1.88	9.5
Perlin	8.19	7.69	6.5

Podemos observar na Tabela I que todos os programas executados no algoritmo apresentaram uma melhora no tempo de execução, tendo sido compilados com o conjunto de otimizações gerado pelo algoritmo VND.

Nota-se que para programas que utilizam um tempo de execução muito pequeno, desde a solução inicial, a diferença na porcentagem do resultado final pode variar bastante.

Tabela II
Resultados Parciais na execução do algoritmo sobre o programa almabench

Solução	Nível de Vizinhança	Tempo de Execução
Inicial (O3)	-	10.69
Solução Gerada	1	10.11
Solução Gerada	1	10.05
Solução Gerada	1	10.05
Solução Gerada	2	10.04
Solução Gerada	1	10.04
Solução Gerada	2	9.55
Solução Gerada	1	9.55
Solução Gerada	2	9.55
Solução Gerada	3	9.55

A Tabela II exibe a evolução dos resultados parciais na execução do algoritmo sobre o programa almabench, podemos observar com a evolução dos resultados que sempre que a melhor solução é atualizada, o nível de vizinhança retorna para 1, como já era esperado segundo a definição da heurística. E por fim, mesmo com o nível de vizinhança sendo incrementado, a melhor solução não é atualizada, e por isso o algoritmo atinge o nível máximo de vinhança, neste caso estipulado como 3, atendendo o critério de parada.

É possivel concluir com a análise dos resultados, que nem todas as otimizações efetivamente melhoram o tempo de execução de um determinado programa, pois em todos os casos testados, o conjunto de otimizações O3 é superado pelos conjuntos gerados pelo algoritmo desenvolvido no quesito tempo.

Referências

[1] Hansen P., Mladenovic N. A Tutorial Variable Neighborhood Search; Charpter 8.3: VARIABLE NEIGHBORHOOD DESCENT – July, 2003. pp. 215-219.