Algoritmos para Dimensionamento e Empacotamento de Grafos Acíclicos Dirigidos

Hilda Helena Alves[1] , Marcos Alves[1] , Ricardo Santos[2]
[1]Engenharia de Computação
Universidade Católica Dom Bosco
hilda.ucdb@gmail.com, malves1.br@gmail.com
[2]Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
ricardo@facom.ufms.br

Abstract. *In this work we present algorithms for sizing Directed Acyclic Graphs (DAGs) of programs. DAGs are data structures which can be used by compilers tools to assign hardware resources (funcional units and registers) to operations of a program. Current modern microprocessors have more than one core and several processing elements inside the chip. In this context, a compiler matches DAGs to the available hardware resources efficiently. One approach to match DAGs to hardware resources is to transform DAGs in rectangles and hardware resources into strips so that the problem is to pack rectangles into strips. We have performed experiments showing that transforming DAGs and adopting packing algorithms is a viable mechanism to match DAGs into hardware resources.*
KEYWORDS. *DAGs-Packing, Strip Packing, Heuristics.*

1 Introdução

Desde sua concepção, métodos de Pesquisa Operacional (PO) têm sido largamente aplicados com o intuito de obter soluções para problemas ligados à tomada de decisões em diversas áreas do conhecimento. Nessas áreas, há aplicações que possuem diversas variáveis e, com isso, o entendimento do problema e o processo de tomada de decisão tornam-se extremamente complexos se realizados manualmente. Por outro lado, a representação do problema através de modelos matemáticos e, por conseguinte, a resolução através de técnicas de Pesquisa Operacional pode reduzir essa complexidade e tornar o processo de resolução bem mais eficiente.

A constante evolução de desempenho dos microprocessadores tem sido uma das maiores características da área de arquitetura de computadores [Hennesy (2004)]. Mesmo com essa característica, há de se notar a necessidade de técnicas de compilação que procuram sobrepor limitações de desempenho dos programas. Dentre essas técnicas, destaca-se o uso de otimizações avançadas sobre o código de um programa a fim de prepará-lo para extrair o máximo desempenho de uma máquina [Aho (2007)]. Essas otimizações são implementas em um software compilador que, além da possibilidade de agrupar algoritmos de otimizações para serem executados sobre o código de um programa, traduzem as declarações do programa em instruções para serem executadas sobre um processador alvo.

Diante desse contexto, este trabalho tem implicações diretas na área de compiladores pois objetiva preparar Grafos Acíclicos Dirigidos (DAGs) de operações de programas para serem eficientemente utilizados pelo compilador. Especificamente, este trabalho propõe a utilização de heurísticas de empacotamento para determinação dos recursos de hardware necessários para executar as operações de um DAG. Propõe-se uma técnica para converter DAGs em retângulos e, posteriormente, utiliza-se algoritmos de empacotamento visando determinar uma área mínima em uma faixa (*strip*) a ser ocupada por esses retângulos. Essa área mínima corresponde ao número de recursos de hardware que são necessários para executar os DAGs. Uma análise preliminar realizada em [Santos (2007)] apresenta o Problema de Empacotamento de DAGs (PED) como uma variação do Problema de Empacotamento por Faixas (*Strip Packing*) [Baker (1980), Lee (1999), Martello (2003)] e propõe heurísticas que utilizam diretamente os DAGs para determinar o número de recursos de hardware necessários para execução de um programa. Os experimentos apresentados em [Santos (2007)] foram aplicados sobre programas de benchmarks bem conhecidos da área de Arquitetura de Computadores e Compiladores e a configuração dos recursos de hardware foi baseada na arquitetura de processador denominada 2D-VLIW [Santos (2006)b].

Este artigo apresenta a seguinte organização: a Seção 2 apresenta o Problema de Empacotamento de DAGs. A Seção 3 é divida em três Subseções: a Subseção 3.1 detalha o algoritmo para determinar a área de um DAG e as Subseções 3.2 e 3.3 apresentam as Heurísticas First Fit e Best Fit para o PED, respectivamente. Os resultados dos experimentos são apresentados na Seção 4. Por fim, na Seção 5 são apresentados as conclusão referentes ao trabalho.

2 Problema de Empacotamento do DAGs

DAGs são grafos acíclicos direcionados que representam a estrutura de dependência entre operações de um programa, no qual os vértices correspondem às

operações e as arestas indicam a existência de uma dependência entre as operações. A Figura 1 exibe o mapeamento de operações em DAGs.

O PED é apresentado como uma variação do Problema de Empacotamento por Faixas (Strip Packing) [Baker (1980), Lee (1999), Martello (2003)], onde os elementos a serem empacotados são considerados retângulos. Aqui considera-se a mesma idéia, os DAGs são os itens a serem empacotados sobre um conjunto de elementos de processamento denominado unidades funcionais (UFs), representados como um grafo base [Santos (2007)]. Sobre o grafo base é possível empacotar diferentes tipos de DAGs. O objetivo do PED é então a determinação do número mínimo de elementos de processamento para executarem um conjunto de operações agrupadas através do DAG.

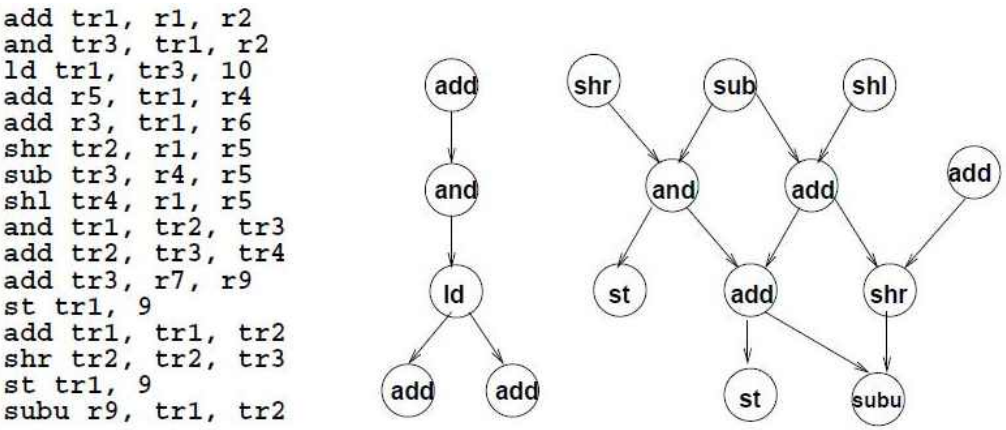

Figura 1. Exemplo do mapeamento de operações em DAGs.

Dependendo da configuração do hardware (processador) adotado, o grafo base pode ter uma topologia de interconexão de nós bastante complexa. Por exemplo, na arquitetura 2D-VLIW utilizada nos experimentos de [Santos (2007)], o grafo base representa uma matriz de UFs. As Figuras 2 e 3 apresentam um exemplo de empacotamento dos DAGs A, B e C em um grafo base representado como uma matriz de UFs..

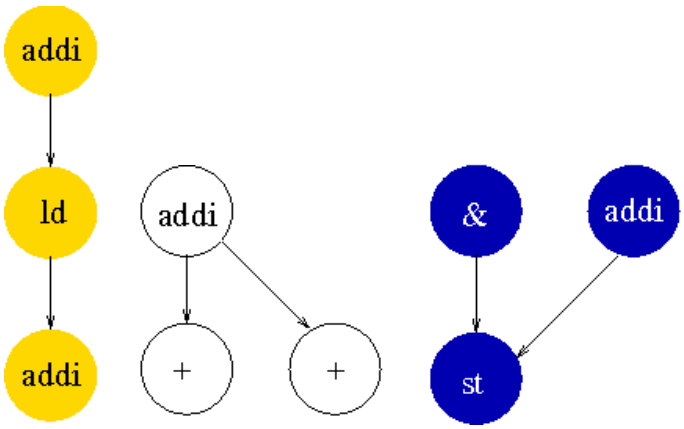

Figura 2. DAGs A, B e C.

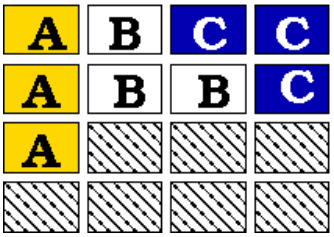

Figura 3. Matriz de UFs necessária para empacotar os DAGs da Figura 2.

3 Algoritmo Para o Problema de Empacotamento de DAGs

Esta seção apresenta a forma como obtém-se a área de um DAG afim de determinar a área necessária para empacotá-lo. As heurísticas clássicas de empacotamento First-Fit e Best-Fit são utilizadas como algoritmo de empacotamento para determinar um grafo base suficiente para os DAGs. Destaca-se aqui que os DAGs são representados como retângulos e, portanto, deve-se determnar a largura (base) e altura de cada DAG.

3.1 Determinação da Área do DAG

As arestas de um DAG possuem pesos que indicam o número de ciclos de clock (tempo) que devem ser obedecidos antes de executar uma próxima operação (nó) do DAG. Este peso afeta diretamente na determinação da altura do DAG uma vez que arestas com peso maior que 1 implicam em DAGs de maior altura.

A altura é determinada pelo maior caminho da raiz até a folha, ressaltando que para obter esse valor o peso de cada aresta deve ser computado juntamento com o peso do vértice folha. A largura de um DAG é calculada como o maior número de arestas que atravessam um mesmo nível de um DAG. Os níveis são dados percorrendo o DAG a partir da folha mais distante no sentido da raiz, de forma que a folha esteja em um nível k e seus antecessores em um nível $k-1$. Nota-se que este percurso não é trivial, pois um DAG pode possuir arestas e nós de diferentes pesos, arestas com diferentes números de filhos e atravessando mais de um nível.

É importante destacar aqui uma característica da largura em relação à configuração do hardware (processador). Existem duas situações no qual a largura de um DAG deve sofrer uma penalidade:

- Caso um nó tenha tenha mais de k filhos, para k=número de interconexões de uma UF.
- A largura do DAG seja maior que m, para m=número de UFs paralelas de um processador.

Em relação a primeira situação, a penalidade deve ser aplicada devido o modelo de interconexão entre as unidades funcionais, de forma que uma UF na linha i pode enviar seus resultados para outras k UFs na linha $i+1$[Santos (2007)]. Já em relação à segunda, a penalidade se dá pelo fato de existir apenas m UFs disponíveis simultaneamente. Para simplificar o entendimento do cálculo da área, a largura de um DAG pode ser interpretada como a base de um retângulo, logo, tendo base e altura de um DAG podemos visualizá-lo como um retângulo, afinal suas dimensões são conhecidas. Assim sendo, a área do DAG é obtida através da multiplicação da

altura pela largura. A Figura 4 apresenta os retângulos obtidos a partir dos DAGs da Figura 2. Neste exemplo, considera-se que as arestas e os vértices, para os três DAGs, possuem peso 1. Assim temos as dimensões obtidas por meio de $altura \times largura$:

- Dimensão DAG A=3=3 × 1;
- Dimensão DAG B=4=2 × 2;
- Dimensão DAG C=4=2 × 2;

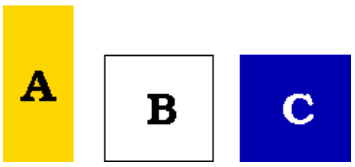

Figura 4. Exemplo de retângulos formados a partir dos DAGs da Figura 2.

3.2 Heurística First Fit para o PED

Tendo o conjunto de itens, ou seja, os retângulos formados a partir dos DAGs, a heurística First Fit cria um nível, empacotando itens ordenados em ordem decrescente pela altura até que um deles não caiba mais no nível. Quando não há espaço em um nível para um item, um novo nível é criado e o item é empacotado neste. Quando há dois ou mais níveis, o item é empacotado no primeiro nível onde há espaço suficiente para o empacotamento.

Pela heurística, o nível possui largura fixa e o início de um nível coincide com o topo do item mais alto do nível anterior. No contexto do PED, a altura de cada nível é fixa, pois um hardware (processador) possui um número fixo de UFs disponíveis. A largura do strip também é fixa pelo mesmo motivo da altura. Essas características da largura são assim estabelecidas, pois representam os recursos de hardware disponíveis.

3.3 Heurística Best Fit para o PED

De modo similar ao First Fit, a heurística Best Fit também empacota um conjunto de itens, formado pelos DAGs de um programa. O Best Fit possui uma particularidade que o difere do First Fit que é a procura pelo melhor nível para empacotar cada item. O nível ideal para o empacotamento de um item é aquele onde após o empacotamento do item o espaço restante seja o menor possível. O Best Fit também respeita as características do PED em relação à altura e largura.

4 Experimentos e Resultados

Para avaliar a implementação, foram realizados experimentos considerando DAGs obtidos a partir dos núcleos de quatro programas como mostra a Tabela 1. Núcleos são as regiões (funções, métodos e/ou procedimentos) mais utilizadas de um programa. Esses núcleos geralmente são os responsáveis pela maior parte do tempo de execução de um programa. Assim, um compilador deve tomar especial atenção sobre essas regiões durante o processo de compilação.

A Tabela 1 apresenta o número de níveis obtidos pelo First-Fit e pelo Best-Fit para empacotar todo o conjunto de DAGs de cada núcleo. É possível comparar os resultados obtidos por First-Fit e Best-Fit com as heurísticas apresentadas

em [Santos (2007)]. Nesse caso, há necessidade de converter os resultados obtidos (grafos base) pelas heurísticas de [Santos (2007)] para níveis. Assim, a coluna *Packing* apresenta os resultados obtidos (em níveis) pela heurística *Packing*. A Tabela apresenta ainda o número de DAGs existente em cada núcleo.

Programa	Núcleo	First-Fit	Best-Fit	*Packing*	Nº DAGs
256.bzip2	_getRLEpair_2	4	4	8	23
	_spec_getc_2	2	2	14	10
	_spec_getc_13	1	1	4	7
197.parse	_insert_dict_2	2	2	4	12
	_rabridged_lookup_2	2	2	4	18
	_rabridged_lookup_16	2	2	4	16
	_xalloc_2	3	3	6	9
175.vpr	_my_strtok_2	3	3	4	16
	_my_strtok_3	1	1	4	4
	_my_strtok_10	1	1	2	3
	_my_strtok_11	2	2	3	9
	_my_strtok_13	2	2	3	11
181.mcf	_primal_iminus_2	4	4	9	46
	_primal_iminus_40	1	1	4	3
	_read_min_30	96	38	224	285

Tabela 1. Resultados dos algoritmos.

O número de níveis necessário para o empacotamento depedende muito da configuração de cada DAG. Isso pode ser facilmente observado comparando os resultados do núcleo _getRLEpair_2 do programa 256.bzip2 com o núcleo _primal_iminus_2 do programa 181.mcf. Apesar do número de níveis ser o mesmo para ambos os núcleos, nota-se diferença na quantidade de DAGs existente em cada núcleo. Outro fato interessante de ser observado é que a estratégia de dimensionar os DAGs em retângulos e aplicar heurísticas de empacotamento apresenta melhores resultados do que a aplicação de heurísticas de *matching* diretamente a partir dos DAGs. Isso pode ser observado quando compara-se os resultados obtidos por First-Fit e Best-Fit com os resultados retornados pela heurística *Packing* apresentada em [Santos (2007)].

O algoritmo Best-Fit apresentou um bom resultado quando comparado em First-Fit para casos onde a quantidade de DAGs é maior. Pode-se notar uma diferença significativa entre os valores obtidos pelos dois algoritmos (terceira e quarta coluna) no programa 181.mcf, nos seus três últimos núcleos. No contexto deste trabalho, essa redução é de considerável importância, uma vez que reduz o número de UFs (grafos base) para o empacotamento de DAGs minimizando os recursos disponibilizados pelo processador e, consequentemente, maximizando o desempenho.

5 Conclusões

Este trabalho apresentou uma estratégia para determinação da área de Grafos Acíclicos Dirigidos (DAGs)que são usados como itens a serem empacotados por algoritmos clássicos de empacotamento. Este empacotamento tem como base o Problema de Empacotamento por Faixas (Strip Packing) e visa a determinação do número

mínimo de Unidades Funcionais necessárias para empacotar DAGs de programas. O algoritmo proposto calcula a área de um DAG a fim de aplicar um algoritmo de empacotamento objetivando assim determinar a área necessária no strip. Para a realização desse cálculo o algoritmo redimensiona o DAG como um retângulo e, assim, calcula sua área a partir da lagura (base) e algura.

Esta proposta apresenta considerável importância para a etapa de geração de código, mais especificamente para a etapa denominada escalonamento de instruções em compiladores que utilizam DAGs como estruturas de dados para representação de programas. Um correto dimensionamento dos DAGs e, por conseqüência, a aplicação de algoritmos de empacotamento, possibilitarão um escalonamento mais eficiente, uma vez que determinam o número mínimo de matrizes de unidades funcionais a serem utilizados. Não foram encontrados trabalhos, na literatura da área, que apresentam outras técnicas baseadas em empacotamento de DAGs visando auxiliar o processo de compilação de programas.

Referências

Aho, A. V., Lam, M. S., Sethi, R., and Ullman., J. D. (2007). Compilers - Principles, Techniques, & Tools. Addison Wesley, 2 edition.

Baker, B. S., Coffman, E. G., and Rivest., R. L. (1980). Orthogonal Packing in Two-Dimensions. SIAM Journal on Computing, 9(4):846-855.

Hennesy, J. L. and Patterson., D. A. (2004). Computer Organization and Architecture: A Hardware/Software Interface. Addison Wesley, 2 edition.

Lee, H. F. and Sewell., E. C. (1999). The Strip-Packing Problem for a Boat Manufacturing Firm. IIE Transactions, 31(7):639-651.

Markoff, J. (May 2004). Intel's Big Shift After Hitting Technical Wall. The New York Times.

Martello, S., Monaci, M., and Vigo., D. (2003). An Exact Approach to the Strip-Packing Problem. Journal on Computing, 15(3):310-319.

Santos, R. (2007). 2D-VLIW: Uma Arquitetura de Processador Baseada na Geometria da Computação. PhD thesis, Instituto de Computação - Universidade Estadual de Campinas, Campinas - São Paulo - Brasil.

Santos, R., Azevedo, R., and Araujo, G. (2006a). Exploiting Reconfiguration Techniques: The 2D-VLIW Approach. In *Proceedings of the* 13^{th} *IEEE International Reconfigurable Architectures Workshop (RAW)*, Rhodes Island-Greece, 2006. IEEE Computer Society.

Santos, R., Azevedo, R., and Araujo, G. (2006b). The 2D-VLIW Architecture. Technical Report IC-06-006, Institute of Computing.

Santos, R., Azevedo, R., and Oliveira, R. M. (2007). A DAGs-Packing Heuristic for a High Performance Processor Architecture. In *Proceedings of the* 39^{th} *Brazilian Symposium of Operational Research*, Fortaleza-Brazil, 2007. SOBRAPO.