Alocação de Registradores a Dado de Tamanho Variável e Escalonamento de Instruções

Mariza Andrade da Silva Bigonha – DCC/UFMG

José Lucas Mourão Rangel Netto – PUC/RJ

I Simpósio Brasileiro de Linguagens de Programação

Setembro/1996

Roteiro da Apresentação

I - Motivação

II - Problemas da Geração e Otimização de Código

- Alocação de Registradores
- Escalonamento de Instruções
- Interdependência entre o Escalonamento e Alocação

III - Generalização dos Algoritmos de Escalonamento e Alocação

- Pares de Registradores

V - Ambiente da Estratégia

VI - Conclusões

I - Motivações

Arquiteturas Superescalares

- Segunda Geração de Arquiteturas RISC
- Processadores com várias unidades funcionais.
- Emissão de mais de uma instrução por ciclo
- Implementam a maior parte das operações de uma única forma.
- Expõem ao gerador e otimizador a estrutura do *pipeline* e custos das unidades funcionais.

Necessidade de Otimização

- Otimização é indispensável para usufruir de todas as vantagens das características da arquitetura.

- O fato de possuir um grande número de registradores impõe ao compilador a tarefa de usá-los eficientemente.

- Algoritmos de escalonamento são utilizados para reduzir os atrasos atribuídos ao tempo de execução no código compilado.

II - Problemas da Geração e Otimização

- Redirecionamento auxiliado por computador.
  Linguagem de Descrição de Arquitetura.

- Alocação de Registradores.
  Pares de Registradores

- Escalonamento de Instruções.
  - Atrasos causados por dependências de dados.
  - Atrasos causados pelas instruções de desvios.
  - Presença de restrições impostas pelos processadores *pipelines*, como por exemplo, a presença ou não de *interlocks* por *hardware*.

- Interdependência entre a alocação de registradores e o escalonamento de instruções.

• Alocação de Registradores

- Pseudo-registradores criados para conter valores intermediários de ex-pressões.
- Número ilimitado de pseudo-registradores.
- Não existe dependência de *hardware*

alocação de registradores

$\Downarrow$

pseudo-registradores $\rightarrow$ registradores físicos

- A alocação é considerada ótima se as variáveis permanecem nos reg-istradores durante todo seu ciclo de vida.
- Alocação adequada de registradores reduz o número de referência à memória.

Grafo de Interferência

Grafo de Interferência: $G_r = (V_r, E_r)$

- Todo vértice $v \in V_r$ corresponde a um intervalo distinto do programa no qual a definição da variável está viva.

- Existe uma aresta não dirigida $(u, v) \in E_r$, se uma definição está viva em u e é necessária em v.

Método mais utilizado para Alocação

- Coloração de grafos (Chaitin,1982; Briggs, 1989).
  - Coloração de grafos modela o problema de alocação de registradore como um grafo de interferência.
  - Atribui cores diferentes para nodos que interferem.
  - Colorir o grafo $\rightarrow$ atribuir registradores físicos aos pseudo-registrad

Método de Chaitin

- O alocador de registradores considera todos os *pseudo-registradores* locais e globais.

- *Pseudo-registradores* são atribuídos a um registrador físico ou são derramados para a memória durante todo o tempo de execução do bloco.

- A alocação de registradores é efetuada em quatro fases.

  – Constrói-se o grafo de interferência.
  – Combinam-se dois *pseudo-registradores*.
  – Colore-se o grafo.
  – Efetua-se o derramamento.

- O método de Chaitin interage na construção do grafo combinando *pseudo-registradores* e derramando até que seja possível colorir o grafo

Para combinar dois *pseudo-registradores* as seguintes condições devem ser satisfeitas:

1. Existe um *move* entre o último uso de um registrador e a definição de outro.

2. Os dois registradores não se interferem.

3. Os vértices correspondendo à combinação dos *pseudo-registradores* não possuem restrições.

Colore-se, primeiramente, vértices com restrições, obedecendo uma ordem decrescente de prioridades, depois colore-se os vértices sem restrições.

A prioridade de cada vértice p é dada por:

$$priority_p = \frac{regcost_p}{degree_p}$$

$$regcost_p = \sum_{r \in refs_p} opcost_r * freq_b$$

$opcost_r \rightarrow$ ciclos necessárias para uma operação de *load* ou *store*.

$b \rightarrow$ bloco básico contendo a referência r.

$freq_b \rightarrow$ quantas vezes b é executado.

$refs_p \rightarrow$ conjunto de todas referências a p.

$degree_p \rightarrow$ vizinhos de p no grafo de interferência.

$regcost_p \rightarrow$ custo, em ciclos de máquina, de forçar p à memória.

Principais características da abordagem de Chaitin

- Uniforme.
- Sistemática.

Desvantagem

- Um pseudo-registrador é atribuido a um registrador físico ou derramado por todo seu tempo de vida.

• Escalonamento de Instruções

Escalonamento de Instruções

Técnica de *software* que rearranja seqüências de código durante a compilação com o objetivo de reduzir possíveis atrasos de execução

- Blocos Básicos.
- Grafo Acíclico Dirigido (DAG).
- Dependências de Dados.
- Algoritmos de Escalonamento.

Blocos básicos

sequência de comandos consecutivos no qual o fluxo de controle entra no seu início e o deixa no final sem interrupções ou possibilidades de desvios exceto em seu final.

Grafo Acíclico Dirigido (DAGs)

O grafo de escalonamento: $G_s = (V_s, E_s)$

- Todo vértice $u \in V_s$ corresponde a uma instrução em uma descriçã do programa baseada em registradores.

- Existe uma aresta dirigida $(u, v) \in E_s$, de u para v, se u deve se executado antes de v.

  1. Existe uma dependência de dados de v em u.

  2. Existe uma dependência de controle de u para v.

  3. Existe uma restrição de recurso de máquina que impõe a precedênci de u sobre v.

Outras propriedades do grafo de escalonamento

- Arestas são rotuladas com a latência de operação existente entre a aresta de origem e os vértices destinos.
- A *raiz* do DAG.
- Uma *folha*.
- Uma *aresta (u,v) com rótulo l*.
- O DAG é ligado por um encadeamento especial que representa a ordem inicial das instruções dentro do bloco básico.
  - O encadeamento é construído percorrendo o bloco básico da última até a primeira instrução, anotando cada definição ou uso de um operando, e relacionando os usos e as definições que devem precedê-los.

Dependências de Dados

- Dependências verdadeiras
- Dependências falsas
  - anti-dependência
  - dependência de saída.

1. Dependência verdadeira ou de fluxo de dados→ registrador definido em u é usado em v.

2. Anti-dependência→ registrador usado em u é redefinido em v e destrói o valor usado em u.

3. Dependência de saída → registrador definido em u é redefinido em v destruindo o valor definido anteriormente em u.

Algoritmos de Escalonamento

- Abordagem mais utilizada:
  - Lista de escalonamento.
  - Uso de heurísticas - auxiliam atribuir prioridades as instruções.

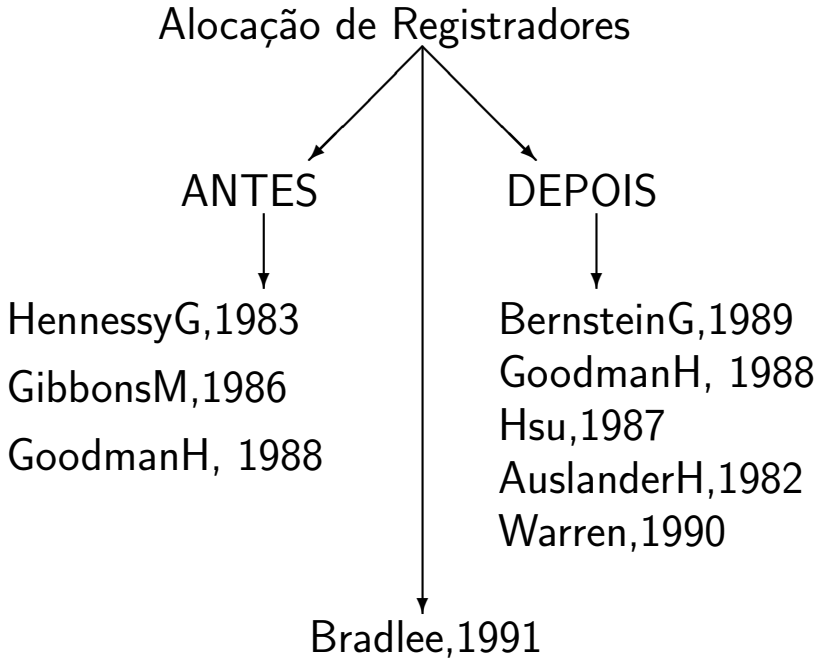
Heurísticas mais utilizadas

$\Downarrow$

- Distância máxima.
- Vértices possuindo mais sucessores.
- Vértices possuindo o maior tempo de latência em relação aos sucessores.

Escalonamento X Alocação

Soluções Existentes



Hennessy e Gross, 1983

1. Alocação de registradores.

2. Escalonamento de instruções.

Utiliza duas listas:

Lista ready: instruções aptas para escalonar.

Lista Leader: predecessores foram escalonados mas se escalonadas causam
dependências de dados.

Heurísticas

- Distância máxima, se não for a única, então,
- Instruções com maior número de sucessores, se não for a única, então
- Instruções com maior latência, se não for a única, então,
- Escolhe arbitrariamente no conjunto de instruções.

Escalonador - $O(n^4)$, n = número de nodos no código DAG.

Gibbons e Muchnick, 1986 - POSTPAS

1. Alocação de registradores
2. Escalonamento de instruções

Heurísticas

- Instruções com maior latência, se não for a única, então,
- Instruções com maior número de sucessores, se não for a única, então,
- Distância máxima.

Escalonador - $O(n^2)$, n = número de nodos no código DAG.

Goodman e Hsu, 1988

1. Alocação de registradores.
2. Escalonamento de instruções.

Idéia básica: alocação de registradores dirigida por DAG. Equilibrar a altura de um DAG para obter um bom escalonamento de instruções

DAG

Largura - número máximo de nodos mutualmente independentes que necessitam um registrador.

Altura - comprimento de seu caminho mais longo.

Goodman e Hsu, 1988 - IPS

1. Escalonamento de instruções.
2. Alocação de registradores.

Idéia básica: gerenciar o número de registradores disponíveis durante o escalonamento de código.

utiliza duas técnica: uma para reduzir os atrasos no *pipeline* quando há um número suficiente de registradores disponíveis, e a outra, para controlar o uso dos registradores.

Problemas:

- Efetua alocação global de registradores manualmente antes de executa IPS.
- Efetua somente alocação local de registradores depois de escalonar.
- Não considera dependências estruturais.

Auslander e Hopkins, 1982 – 801

1. Escalonamento de instruções.
2. Alocação de registradores.
3. Escalonamento de instruções.

Warren, 1990 – RS/6000

- Escalonador baseado no método de Auslander e Hopkins.
- Acrescenta heurísticas que limitam o efeito do escalonamento de instruções sobre o alocador de registradores.
- Diminui prioridade de *loads* quando a necessidade por registradores aumenta.

Bradlee, Eggers e Henry, 1991 – RASE

Distância máxima - primeira heurística

Grau elevado de acoplamento

- Preschedule.
- Alocação global de registradores.
- Finalschedule.

Interdependência entre Escalonamento e Alocação

- Alocação de registradores ANTES.

Problema:

pode introduzir dependências atribuindo o mesmo registrador físico para instruções sem nenhuma relação

Desvantagem:

se os registradores forem referenciados no mesmo bloco básico, o escalonador não pode superpor operações que os usam.

pr3	←	pr1 * pr1
store	pr3	
pr4	←	pr1 * pr2
store	pr4	

			1^a escolha	2^a escolha
pr1		pr1	r1	r1
pr2		pr2	r2	r2
pr3		pr3	r3	r3
pr4		pr4	r3	r4
pr5		pr5	r4	spill

ciclos	instruções	ciclos	instruções
0	r3 ← r1 * r1	0	r3 ← r1 * r1
4	store r3	1	r4 ← r1 * r2
5	r3 ← r1 * r2	4	store r3
9	store r3	5	store r4

Alocação Registradores *antes* Escalonamento Instruções

ciclos	instruções
0	load r1, 12(fp)
1	load r2, 16(fp)
2	load r3, 40(fp)
3	load r4, 44(fp)
4	r1 $\leftarrow$ r1 + r2
5	r2 $\leftarrow$ r3 + r4

ciclos	instruções
0	load r1, 12(fp)
1	load r2, 16(fp)
2	load r3, 40(fp)
3	r1 $\leftarrow$ r1 + r2
4	load r2, 44(fp)
6	r2 $\leftarrow$ r3 + r2

Dois escalonamentos para *loads* e *adds* independentes

III - Generalização do Algoritmo de Alocação e Escalonamento

Objetivos da Estratégia

- Encontrar um algoritmo ótimo tal que ele faça uso de um número mínimo de registradores físicos
- Que não derrame valores ainda vivos para a memória.
- Cujo grafo de escalonamento não possua dependências falsas.

O ponto principal da estratégia proposta é o desenvolvimento de um alocador de registradores que não restringe a ação do escalonador.

Suposições Básicas

- Grande número de registradores de propósito geral, $r_1, r_2, \cdots, r_n$,
- Células de memória $m_1, m_2, \cdots, m_n$,
- Coleção de unidades funcionais $\rightarrow$ cada uma pode executar uma instrução a cada ciclo de máquina,
- As operações:
  - Operações de carga: $(m_i) \rightarrow (r_j)$.
  - Operações de armazenamento: $(r_i) \rightarrow (m_j)$.
  - Operações aritmeticas: $A((r_{ij}), \cdots, r_{ik})) \rightarrow (r_j)$, onde, $k >= 1$.
- Algoritmo baseado em listas de escalonamento.
- Algoritmo baseado no método de Chaitin.
- Grafos utilizados: G_s e G_r.

Exemplo

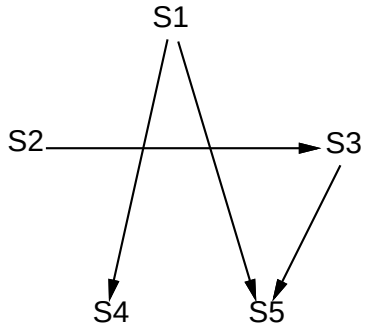
```
x  :=  a[i]
y  :=  z + z
z  :=  x * 5 + z
       (1)
```

```
s1  :=  load z
s2  :=  load i
s3  :=  a[s2]
s4  :=  s1 + s1
s5  :=  s3 * 5 + s1
        (2)
```

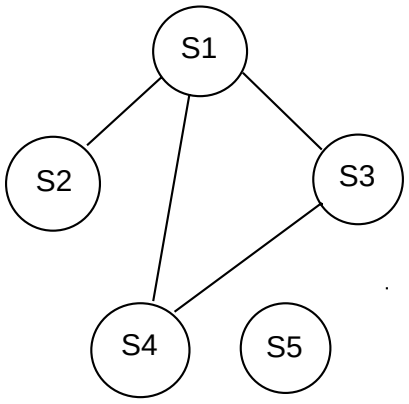
```
    r1  :=  load z
⇒   r2  :=  load i
    r3  :=  a[r2]
⇒   r2  :=  r1 + r1
    r1  :=  r3 * 5 + r1
            (3)
```

$$
\begin{aligned}
\mathsf{s1} &:= \mathsf{load\ z} \\
\mathsf{s2} &:= \mathsf{load\ i} \\
\mathsf{s3} &:= \mathsf{a[s2]} \\
\mathsf{s4} &:= \mathsf{s1} + \mathsf{s1} \\
\mathsf{s5} &:= \mathsf{s3} * 5 + \mathsf{s1}
\end{aligned}
$$

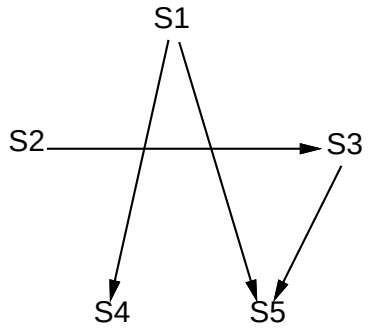
Grafo de Escalonamento G_s



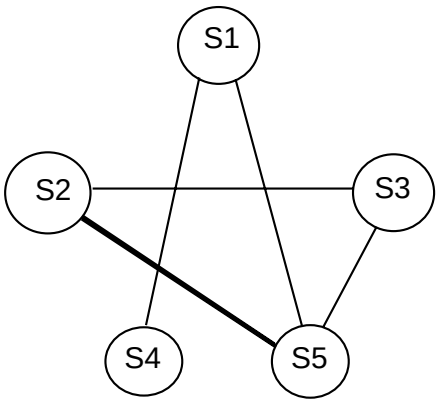
Grafo de Interferência G_r



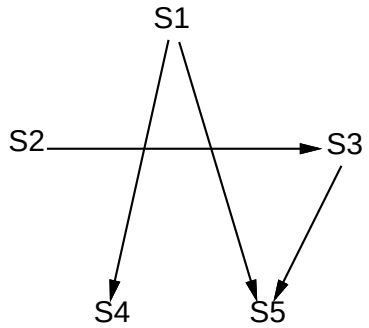
s1 := load z
s2 := load i
s3 := a[s2]
s4 := s1 + s1
s5 := s3 * 5 + s1



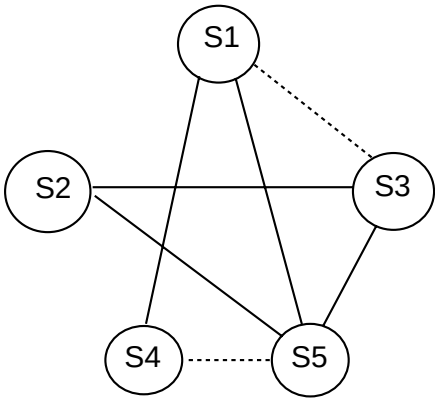
Fecho Transitivo



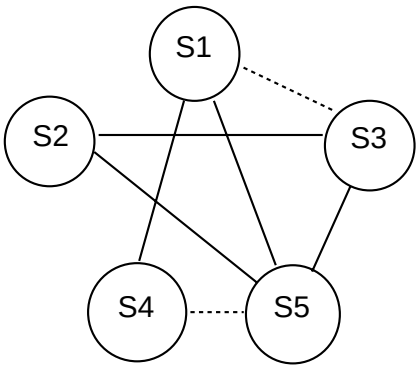
s1 := load z
s2 := load i
s3 := a[s2]
s4 := s1 + s1
s5 := s3 * 5 + s1



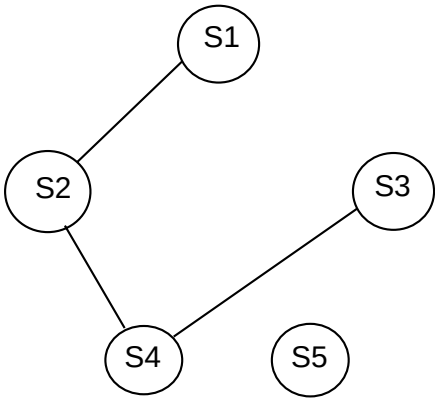
Vértices no Conjunto E_t.



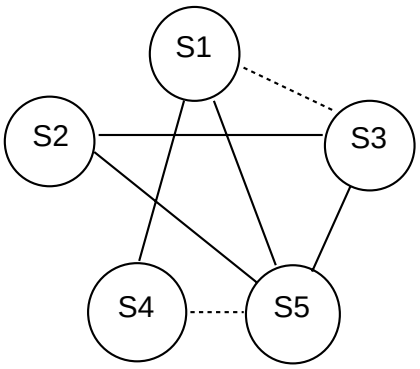
Vértices no Conjunto E_t.



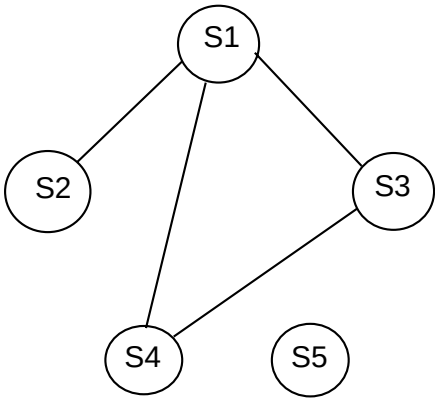
Complemento do Grafo



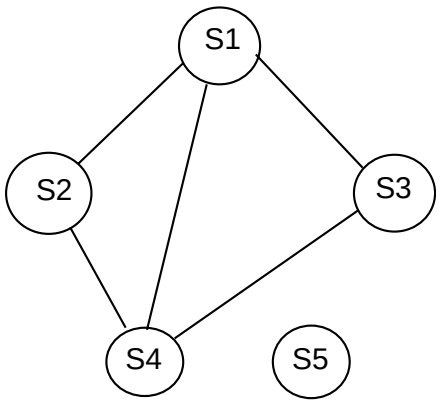
Vértices no Conjunto E_t.



Grafo de Interferência



Grafo de Interferência Paralelizável



Código: Grafo de Interferência Paralelizável

$$
\begin{array}{rlll}
 & \mathsf{r1} & := & \mathsf{load\ z} \\
\Rightarrow & \mathsf{r2} & := & \mathsf{load\ i} \\
 & \mathsf{r2} & := & \mathsf{a[r2]} \\
\Rightarrow & \mathsf{r3} & := & \mathsf{r1} + \mathsf{r1} \\
 & \mathsf{r2} & := & \mathsf{r2} * 5 + \mathsf{r1}
\end{array}
$$

Código: Grafo de Interferência

$$
\begin{array}{rlll}
 & \mathsf{r1} & := & \mathsf{load\ z} \\
\Rightarrow & \mathsf{r2} & := & \mathsf{load\ i} \\
 & \mathsf{r3} & := & \mathsf{a[r2]} \\
\Rightarrow & \mathsf{r2} & := & \mathsf{r1} + \mathsf{r1} \\
 & \mathsf{r1} & := & \mathsf{r3} * 5 + \mathsf{r1}
\end{array}
$$

Derramamento de Código

Alocação de registradores → encontrar um mapeamento de registradores tal que o custo de derramamento seja mínimo.

⇓

Aplicam-se sobre o *grafo de interferência paralelizável* as mesmas heurística usadas tanto na alocação de registradores como no escalonamento de instruções.

Heurísticas → eliminam arestas do grafo.

- Arestas que evitam dependências falsas
  algumas opções de paralelismos são perdidas.

- Arestas de interferência as quais podem ocasionar derramamentos de código.

Regras para Derramamento

- Remova todas as arestas de $E - E_r$ para as quais o escalonamento paralelo de duas instruções possui a menor prioridade para o escalonamento.

- Evite a remoção das arestas em $E_f \cap E_r$. Estas arestas são usadas pelo escalonador e pelo alocador.

Outra abordagem

Função heurística $h \rightarrow$

$$h(v) = cost(v)/degree(v)$$

Pares de Registradores

Como pares de registradores não possuem nehuma relação com o objetivo
do algoritmo proposto, que é manter o paralelismo de instruções evitando
dependências falsas no *grafo de interferência paralelizável*, eles podem se
incorporados neste algoritmo como em qualquer outro esquema.

Continuação: Pares de Registradores

- A necessidade por mais de um registrador para um dado vértice v muda a definição de *degree*.

$$degree_p \rightarrow \textit{vizinhos de } p \textit{ no grafo de interferência.}$$

- Um vértice v é considerado sem restrições se: $degree(v)+need(v)$ for menor que o número de registradores alocáveis

- Um vértice v pode ser removido do *grafo de interferência paralelizáve* durante a fase de simplificação se o predicado *is-safe(v)* é verdadeiro

$$\textit{is-safe(v)} = degree(v) + need(v) \leq r$$

1. $degree(v) =$ soma das necessidades de registradores físicos de seus vizinhos.

2. r representa os registradores físicos disponíveis.

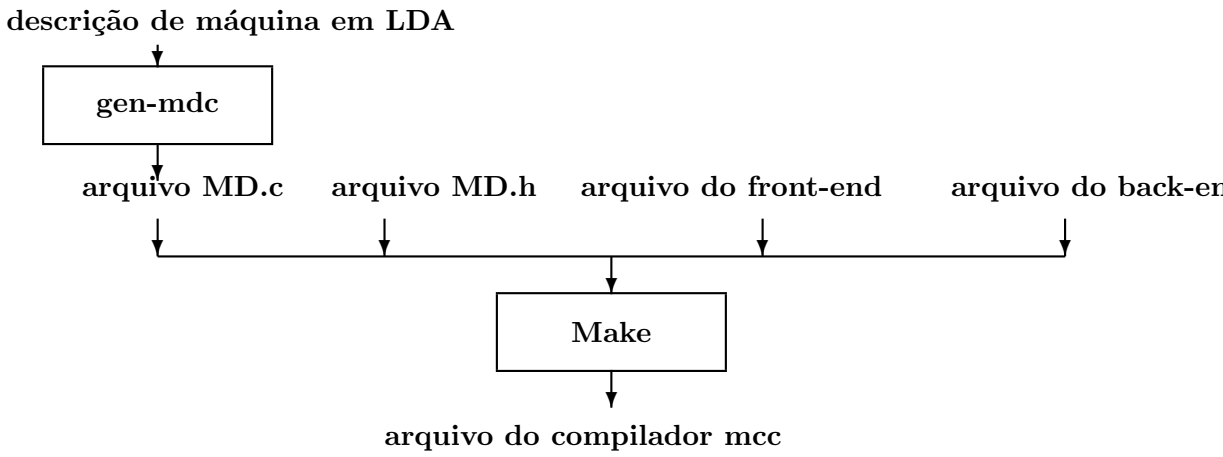
3. $need(v)$ representa o número de registradores físicos necessários à v.

Pontos Positivos desta Abordagem

- O escalonador não é afetado pelo alocador de registradores.
- O grafo de escalonamento continua sendo G_c.
- O grafo para alocar registradores é o *grafo de interferência paralelizável*
- O *grafo de interferência paralelizável estendido* pode ser usado para as duas funções de escalonamento e alocação.
- O algoritmo de escalonamento não precisa ser tão complicado como RASE, podendo inclusive usar o algoritmo proposto por Gibbon cu- jas técnicas heurísticas empregadas são simples, contudo capazes de prover um bom escalonamento.

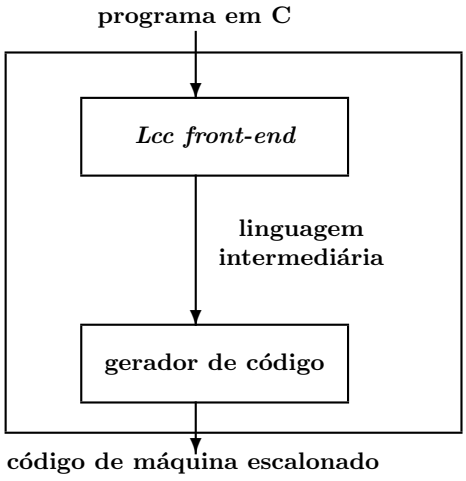
IV - Ambiente da Estratégia

Sistema Gerador de Geradores de Código

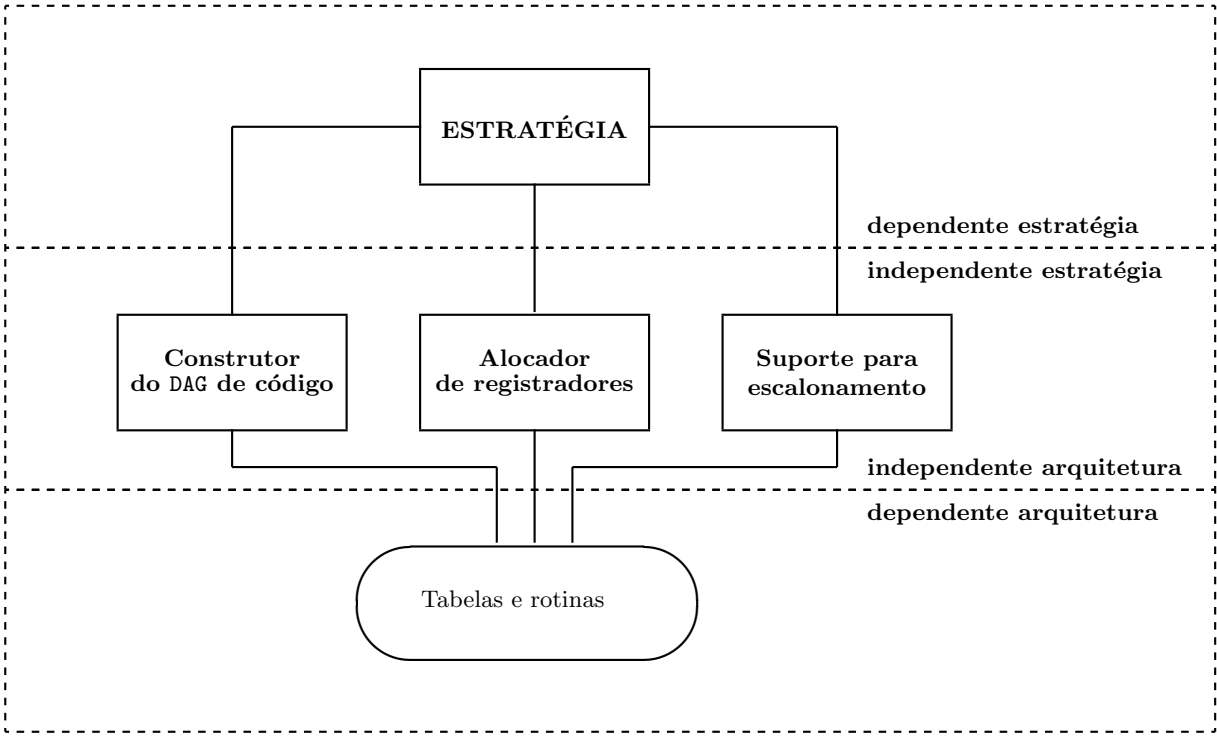


Compilador mcc

- Fase de transformação.
- Fase de construção de blocos básicos.
- Fase de transformações dependentes de arquitetura.
- Fase de casamento de padrões (*match*) e geração de código.
- Fase de estratégia de escalonamento e alocação de registradores.



Estrutura do Módulo Gerador de Código



V - Conclusões

- Identificação de técnicas para aumentar a eficiência do escalonamento de instruções e o nível de comunicação que deve existir entre o alocador e o escalonador de instruções.

- Proposta de modificação do algoritmo de Pinter para tratar o uso de mais de um registrador por variável.

A continuação deste trabalho compreende:

- Implementação e avaliação do algoritmo de alocação de registradores proposto.