Paralelismo em Nível de Instrução

Estagiária PAE:
Maria Lucía Castro Jorge

Paralelismo

- Capacidade de executar mais de uma operação num ciclo de relógio
  - Melhor performance do processamento de dados
- Hoje em dia todos os processadores permitem o processamento de instruções em paralelo

2

Paralelismo em Nível de Instrução

- Chama-se paralelismo em nível de instrução pois se aplica sobre o código de instruções gerado pelo compilador
- O paralelismo é uma forma de otimização de código

3

Velocidade de execução de um programa em um processador paralelo

- Depende de:
  - Paralelismo Potencial do programa
  - Paralelismo disponível no processador
  - Capacidade de extrair o paralelismo
  - Capacidade de achar o melhor escalonamento Paralelo

4

Estratégias para exploração de paralelismo

- Dinâmica (Hardware): descobrir e explorar o paralelismo de instrução durante a execução
- Estática (software): descobrir o paralelismo durante a compilação

5

Restrições para escalonar código

- Dependência de dados
  - Programa otimizado deve produzir os mesmos resultados que programa original
- Dependência de Controle
  - Todas as operações executadas no programa original devem ser executadas no programa otimizado
- Recursos
  - O código não deve reter os recursos do computador por muito tempo

6

Dependência de Dados

- As vezes mais de uma operação pode acessar o mesmo espaço de memória durante a compilação, portanto alguns dados precisarão ser calculados mantendo alguma seqüência para não alterar os resultados

7

Dependencia de Dados

- Tipos de dependência:
  - Verdadeira: ler após escrever
    - Add R1, R2, R3
    - Sub R4, R1, R5
  - Antidependência: escrever após ler
    - Add R1, R2, R3
    - Sub R2, R4, R5
  - Dependência de saída: escrever após escrever
    - Add R1, R2, R3
    - Sub R1, R4, R5

8

Exercício 1

- Considere o seguinte trecho de código

  a = 1;
  **p = 2;*
  x = a;

  Quais tipos de dependência de dados há?

  **Cuidado com os sinônimos de apontadores*

Exercício 2

```
for ( i = 0; i < n; i ++)
   A [2*i] = A [ 2*i +1 ];
```

Dependências ?

```
for ( i = 0; i < n; i ++)
      x = x + A [ 2*i +1 ];
```

Exemplo: como paralelizar?

```
LD  R1, a        // R1 = a
ST  b,R1         // b = R1
LD  R2, c        // R2 = c
ST  d, R2        // d = R2
```

Pouco paralelismo!

11

Exercício em duplas
Para entregar

```
LD R1, a
LD R2, b
ADD R1, R1, R2
LD R2,c
ADD R1, R1, R2
LD R2, d
LD R3, e
ADD R2, R2, R3
ADD R1, R1, R2
```

- Como podemos introduzir o paralelismo?
- Isso teria alguma desvantagem?

12

Exercício

```
LD  R1,a          // R1=a
LD  R2,b          // R2=b
LD  R3,c          // R3=c
LD  R4,d          // R4=d
LD  R5,e          // R5=e
ADD R6, R1, R2    // R6=R1+R2
ADD R7, R4, R5    // R7=R4+R5
ADD R8, R6, R3    // R8=R6+R3
ADD R9, R8, R7    // R9=R8+R7
```

13

Linha de montagem para código com paralelismo

R1=a	R2=b	R3=c	R4=d	R5=e
R6=R1+R2	R7=R4+R5			
R8=R6+R3				
R9=R8+R7				

14

Uso de registradores e paralelismo

- O uso de poucos registradores pode implicar pouco paralelismo no código
- Por outro lado, o uso de muitos registradores pode aumentar o código demais e anular as vantagens da otimização de código

Analise Custo/Beneficio

15

Dependência de Controle

- Uma instrução i_2 é dependente de controle de uma instrução i_1 se o resultado de i_1 determina se i_2 deve ser executado ou não

  if (c) **then** S1 **else** S2
- Abordagem especulativa: dados são calculados com antecipação e aplicados condicionalmente

  if (a = = 0) **then** b= c + d;

  add R3,R4, R5

  cmovz R2, R3, R1

16

Escalonamento de Instruções

- Escalonamento de bloco básico
  - Bloco básico: trecho de código seqüencial onde só existe um ponto de entrada e um de saída
  - Qualquer instrução de desvio encerra um bloco básico
- Escalonamento global de código
  - Bloco global: envolve um conjunto de blocos básicos que podem ter dependência ou não entre eles

17

Escalonamento de Bloco básico

- Problema NP Completo
- Requer técnicas de escalonamento simples já que o bloco contem um numero limitado de operações altamente restritas
- Algoritmo utilizado: escalonamento de lista
  - Requer um grafo de dependência de dados

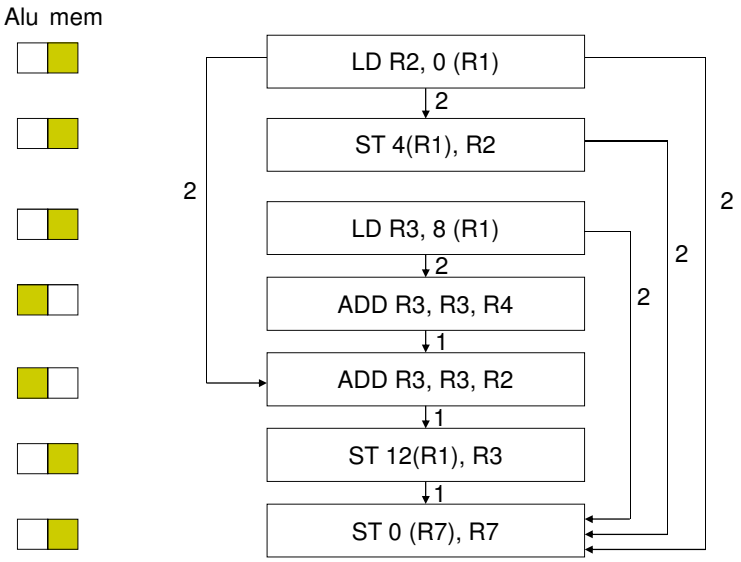
18

Grafo de dependência

- G = (N,E)
  - N (Nós) = Conjunto de operações
  - E (arestas) = restrições de dependência
- Cada operação tem uma tabela de reserva de recurso
- Arestas são rotuladas com o valor do atraso de cada operação (latência)

19

Exemplo Grafo de Dependência



20

Algoritmo de escalonamento de lista

- Grafo é percorrido em ordem topológica priorizada
  - A visita em alguns nós tem prioridade sob outros nós segundo vários critérios (caminho crítico, ordem de aparição das operações, etc.)
  - Os nós com prioridade são escalonados primeiro
- Para cada nó encontra-se o tempo mais curto em que pode ser executado
- Verifica-se a disponibilidade de recursos para a operação

21

Exercício de aplicação do algoritmo

- Aplique o algoritmo de escalonamento de lista de blocos básicos do exemplo anterior mostrando o uso da tabela de recursos

22

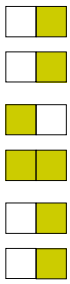
Exercício de aplicação do algoritmo

- Aplique o algoritmo de escalonamento de lista de blocos básicos do exemplo anterior mostrando o uso da tabela de recursos

Escalonamento

	LD R3, 8(R1)
	LD R2, 0(R1)
ADD R3,R3,R4	
ADD R3,R3,R2	ST 4 (R1),R2
	ST 12(R1),R3
	ST 0(R7),R7

Tabela de recursos



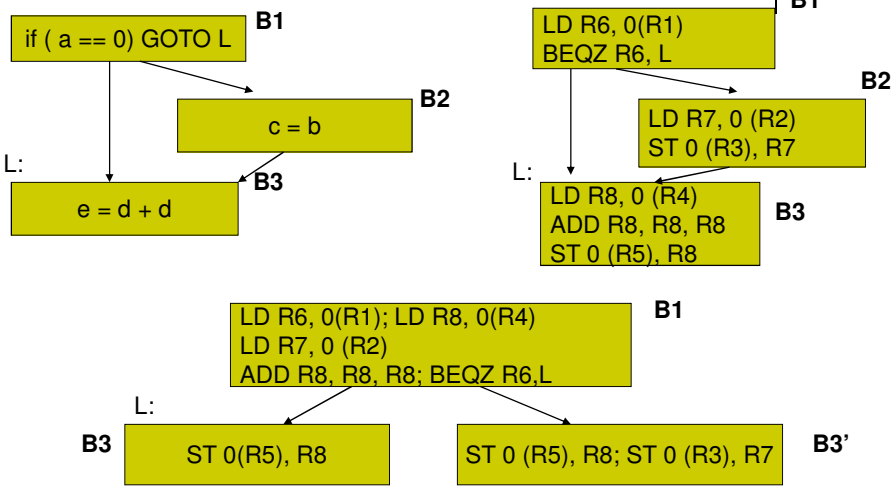
23

Escalonamento Global de código

- Escalonamentos criados pela compactação de blocos básicos tendem a deixar recursos ociosos
  - Estratégias para movimentar instruções entre blocos básicos
  - Deve-se garantir que
    - Todas as instruções originais sejam executadas
    - Instruções especulativas não devem causar efeitos colaterais
    - Tomar cuidado com a dependência de dados

24

Exemplo



25

Software Pipelining

- Técnica de escalonamento para processadores com múltiplas unidades funcionais que busca otimizar o código de um laço explorando o paralelismo entre as distintas instruções dentro do laço

26

Software Pipelining: Exemplo

```
for ( i = 0; i < n; i ++ )
    D[ i ] = A[i] * B[i] + c
```

- Código de maquina: Assumindo que R1, R2, R3, R4 e R5 já foram carregados com os valores de A, B, c, e n-1 respectivamente

```
LD R6, 0 (R1++)
LD R7, 0 (R2++)
MUL R8, R6, R7
ADD R9, R8, R4
ST 0(R3++), R9
```

27

Software Pipelining: Exemplo

- O laço pode ser desdobrado em 4 iterações

```
for ( i = 0; i < n; i +=4 ){
    D[ i ] = A[i] * B[i] + c
    D[ i +1] = A[i+1] * B[i+1] + c
    D[ i +2] = A[i+2] * B[i+2] + c
    D[ i+3] = A[i+3] * B[i+3] + c
    }
```

28

Software Pipelining: Exemplo

Ciclo	j=1	j=2	j= 3	j=4
1	LD			
2	LD			
3	MUL	LD		
4		LD		
5		MUL	LD	
6	ADD		LD	
7			MUL	LD
8	ST	ADD		LD
9				MULT
10		ST	ADD	
11				
12			ST	ADD
13				
14				ST

29

Escalonamento em grafos de dependência acíclicos

- Cada iteração do laço possui um intervalo de iniciação mínima, que é o tempo pelo qual cada iteração segue a outra
- O objetivo com grafos acíclicos é escalonar as operações considerando um limite no intervalo de iniciação mínima

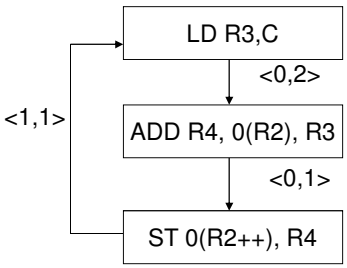
30

Escalonamento em grafos de dependência acíclicos

- As arestas dos grafos estão rotuladas não apenas com o valor de latência mas também com o numero de iteração em que o laço se encontra
- As operações são escalonadas em ordem topológica priorizada usando o escalonamento de lista já conhecido

31

Escalonamento em grafos acíclicos




```
for (i=0; i <N-1; i++)
    A[i]=A[i] +c;
```

32

Loops do across

- Laços cujas iterações compartilham dependências

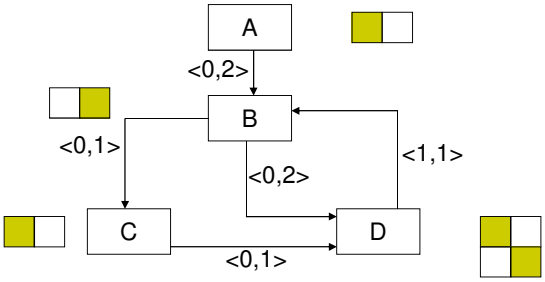
```
for ( i = 0; i <n; i ++ ){
    sum= sum+ A[i];
    B[i] = A[i] * b;
}
```

- Neste caso a ordem seqüencial deve ser realizada

33

Escalonamento em grafos de dependência cíclicos

- O conceito de “ordem topológica” não existe em grafos cíclicos



34

Resumo

- Para executar instruções de modo paralelo é importante analisar tanto as dependências de dados, de controle e de recursos
- O paralelismo pode ser atingido introduzindo novos registradores, mas deve-se fazer uma análise de custo/beneficio
- Dois tipos de escalonamento: básico e global

35

Resumo

- Escalonamentos são feitos seguindo principalmente uma visita de ordem topológica priorizada no grafo de dependências
- O software Pipelining é um conjunto de técnicas para resolver paralelismo em laços, considerando tanto grafos cíclicos quanto acíclicos

36

Referencia

- Aho, A.V.; Lam, M.S; Sethi, R.; Ullman, J.D. (2007). Compiladores: Princípios, técnicas e ferramentas. Cap. 10, pp. 443-480.

37