

Um Simulador Educacional para Análise de Dependências de Dados em Nível de Instrução

Maikel Losekann

URI

maikelosekann@hotmail.com

Guilherme Schievelbein

URI

guilhermeschievelbein@yahoo.com.br

Cristina Paludo Santos

URI

paludo@urisan.tche.br

Alexandre Dos Santos

URI

Ale.roque@gmail.com

ABSTRACT

This paper presents the simulator AD3W developed with the purpose of serve as support for the teaching and learning of topic detection data dependency, widely discussed and used in the disciplines of computer architecture. Its use reduces learning difficulties, facilitating the visualization of the execution of instructions considering the various dependencies between them.

RESUMO

Este artigo apresenta o simulador AD3W, desenvolvido com o intuito de servir como apoio ao ensino e a aprendizagem do tópico de detecção de dependência de dados, amplamente discutido e utilizado nas disciplinas de arquitetura de computadores. A sua utilização permite reduzir as dificuldades de aprendizagem, facilitando a visualização da execução de instruções considerando as diversas dependências existentes entre elas.

Palavras-Chave:

Ferramenta de Apoio ao Ensino, Simulador, Arquitetura de Computadores

1. INTRODUÇÃO

O uso de tecnologias de informação e comunicação no contexto educacional é uma realidade no cenário contemporâneo. As contribuições providas pelo seu uso vinculado ao processo de ensino e de aprendizagem têm sido apontadas por vários estudiosos e educadores ao longo dos últimos anos [1][8].

Direcionando-se, mais especificamente, para a área de arquitetura de computadores, pode-se apontar diversas ferramentas e técnicas já utilizadas para auxílio ao processo de ensino e de aprendizagem de disciplinas relacionadas a esta área[1][3][10]. Dentre elas, os simuladores se destacam por serem ótimas opções para ilustrar conceitos de maneira simples e com um bom nível de interação, promovendo uma aproximação entre os conceitos teóricos e sua aplicação prática [4][7][8].

Em arquiteturas paralelas, por exemplo, o uso de simuladores apresenta-se como uma alternativa promissora para ilustrar de forma interativa as possibilidades de paralelismo em nível de instrução, representando a distribuição feita por um escalonador a fim de prover ganhos de *speedup* e economia de recursos [9].

Acredita-se que o uso de ferramentas gráficas e/ou visuais para abordar tais conceitos encoraja e estimula o estudo nas áreas de processamento paralelo e computação de alto desempenho (PAD/HPC), contribuindo de forma significativa para promover um melhor entendimento da aplicabilidade prática dos mesmos e

maximizando os impactos desejados no processo de ensino e de aprendizagem.

A reflexão acerca dessa questão incitou o desenvolvimento do AD3W – um simulador de cunho educacional para análise de dependências de dados em nível de instrução. A concepção do simulador promove possibilidades de realização de estudos detalhados sobre as propriedades de um sistema e seu funcionamento interno [3], visto que propõem a realização de diversos exercícios que permitem analisar o comportamento da execução de instruções considerando os diferen

tes tipos possíveis de dependência.

Uma descrição mais detalhada dos princípios que regem o desenvolvimento do AD3W é apresentada nas próximas seções como segue. A seção 2 apresenta uma breve contextualização dos conceitos que subsidiaram o desenvolvimento do AD3W, bem como dos aspectos metodológicos envolvidos na concepção do simulador. Já, a Seção 3, apresenta o protótipo desenvolvido e os resultados obtidos com a execução de testes para validação do mesmo. Por fim, a Seção 4 apresenta as considerações finais relativas ao trabalho desenvolvido.

2. CONTEXTUALIZAÇÃO E ASPECTOS METODOLÓGICOS

Para obtenção do paralelismo em nível de instrução a utilização de *pipelines* paralelos em arquiteturas superescalares é fundamental. Três políticas de iniciação de instruções são tratadas para o aprendizado dos *pipelines* [5]:

- Iniciação em Ordem com terminação em Ordem;
- Iniciação em Ordem com terminação fora de Ordem;
- Iniciação fora de Ordem com terminação fora de Ordem.

As informações de dependência de dados são essenciais para detectar iterações de laços que podem ser executadas em paralelo por arquiteturas multiprocessadas e vetoriais [11]. Dependência de dados ocorre quando uma instrução depende do resultado de outra instrução que ainda não foi executada (ainda está na *pipeline*). Este tipo de dependência é originado na natureza sequencial do código em execução, cuja ordem normal é alterada dentro do *pipeline*.

De acordo com [5] a detecção de dependências (*hazard classification*) é organizada em: *data hazards*, *structural hazard* e *control hazard/branching hazards*. A análise de dependências nestes níveis é importante para tratamento de diversos problemas comuns que são abordados na literatura de ensino de arquitetura de computadores e possibilitam diversos algoritmos para

tratamentos específicos como, por exemplo, o algoritmo para renomeação de registradores (*algoritmo de tomazulo*) e algoritmo de predição de desvios (*branch prediction*) [5].

Tais dependências podem ser classificadas da seguinte forma:

- Dependência de dados / *Data hazards*

- WAR (*Write After Read*) – ocorre quando a instrução i+1 tenta escrever um dado antes que a instrução i possa lê-lo;
- RAW (*Read After Write*) – ocorre quando a instrução i+1 tenta ler um dado que a instrução i ainda não atualizou;
- WAW (*Write After Write*) – ocorre quando a instrução i+1 tenta escrever um dado antes que a instrução i atualize o mesmo;

- Dependência de desvios / *Branching hazards*

- DESVIO – instruções a baixo de uma instrução de desvio.

A Figura 1 ilustra alguns exemplos de dependências. Ressalta-se que instruções do tipo RAR (*Read-After-Read*) não são consideradas dependências.

RAW	WAR	WAW
$\underline{A} = B + C$	$A = B + \underline{C}$	$\underline{A} = B + C$
$X = \underline{A} + Y$	$\underline{C} = X + Y$	$\underline{A} = X + Y$

Figura 1. Exemplos de dependências. (Fonte: Autor)

A análise das dependências de dados é fundamental para possibilitar otimizações e detecção de paralelismo implícito em programas sequenciais. Esta análise oferece as informações necessárias para realizar transformações coerentes capazes de proporcionar melhorias de localização em memória, balanceamento de cargas e escalonamento eficiente. [11].

Quanto aos aspectos metodológicos, o simulador proposto trata das 3 políticas descritas anteriormente, ilustrando as diferentes formas de execução por meio de exemplos e exercícios. Além disso, no simulador desenvolvido são tratadas as dependências de dados e os desvios de controle, utilizando como base a lógica dos algoritmos para renomeação de registradores e de predição de desvios.

O simulador AD3W foi desenvolvido com o objetivo de reduzir as dificuldades de aprendizagem sobre o tópico de detecção de dependência de dados, amplamente discutido e utilizado nas disciplinas de arquitetura de computadores.

3. PROTÓTIPO DO SIMULADOR AD3W

O protótipo do simulador proposto foi desenvolvido na linguagem de programação JAVA que oferece suporte a orientação a objetos e recursos robustos para desenvolvimento de interfaces gráficas que facilitam a interação do usuário. Além disso, diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem JAVA é compilada para um *bytecode* [6] que é executado por uma máquina virtual, oferecendo, assim, uma maior portabilidade. O desenvolvimento de simuladores apresenta diversos exemplos de *softwares* desenvolvidos na plataforma JAVA, principalmente os *Applets*. Entre estes aplicativos observamos o *Easy Java Simulations* que provê um mecanismo de criação de simuladores de Física em JAVA [2].

No contexto do AD3W, considera-se que cada instrução tem vários aspectos (atributos) que divergem entre si e estes são

comparados uns com os outros para detectar as dependências de dados. Esta comparação se torna menos complexa quando cada aspecto é definido como um atributo de uma classe. Assim, para a composição de uma instrução *Assembly*, foi estabelecida uma relação entre uma classe e seus atributos como é apresentado na Figura 2.

```

class Instrucao
{
- private id : int
- private nroCiclos : int
- private cicloExe : int
- private cicloSaida : int
- private receptor : String
- private op1 : String
- private op2 : String
- private nop : boolean = false
- private rot : boolean = false
- private desvio : boolean = false
- private executado : boolean = false
- private naJanela : boolean = false
- private foiPrim : boolean = false
- private escrita : boolean = false
- private WARcom [ ] : Integer = new ArrayList <>(): Integer
- private RAWcom [ ] : Integer = new ArrayList <>(): Integer
- private WAWcom [ ] : Integer = new ArrayList <>(): Integer
- private DESVcom [ ] : Integer = new ArrayList <>(): Integer
}
    
```

Figura 2. Classe que compõe os aspectos de uma instrução. (Fonte: Autor)

Para realizar a análise são atribuídos a cada instrução alguns atributos de controle, que são:

- **Id:** identificação de cada instrução, varia de 1 a “n” em que n é o número total de instruções, atribuído de forma crescente a cada instrução.
- **Número de ciclos:** a quantidade de ciclos que cada instrução levará para executar.
- **Número de ciclos executados:** número incrementado a cada ciclo de execução e representa quantos ciclos da instrução foi executado.
- **Ciclo de saída:** qual o ciclo de execução que a instrução irá sair da janela de execução.
- **Operadores:** representam os registradores de cada instrução, o receptor (registrador que recebe o resultado da operação) e os operandos.
- **Variáveis booleanas:** são variáveis que irão representar se a instrução é uma instrução de desvio, de rótulo ou do tipo NOP (*no operation* ou finalização), bem como se a instrução foi executada, foi escrita, se já foi para a execução ou ainda está na janela de instruções.

Além destes valores, também são representadas as instruções que possuem dependências com as demais. Tal representação é feita por meio de uma lista onde cada instrução ocupa uma posição.

Durante a instanciação da classe, diferentes valores podem ser aplicados a cada um dos atributos. Além disso, para ter acesso a uma instrução na lista, basta referenciar sua “id” que resultará na posição ocupada por ela na lista. A seguir a Figura 3 apresenta a lógica utilizada para a análise das instruções, segundo os

princípios do simulador AD3W, detalhando as decisões e aspectos mais relevantes a considerar.

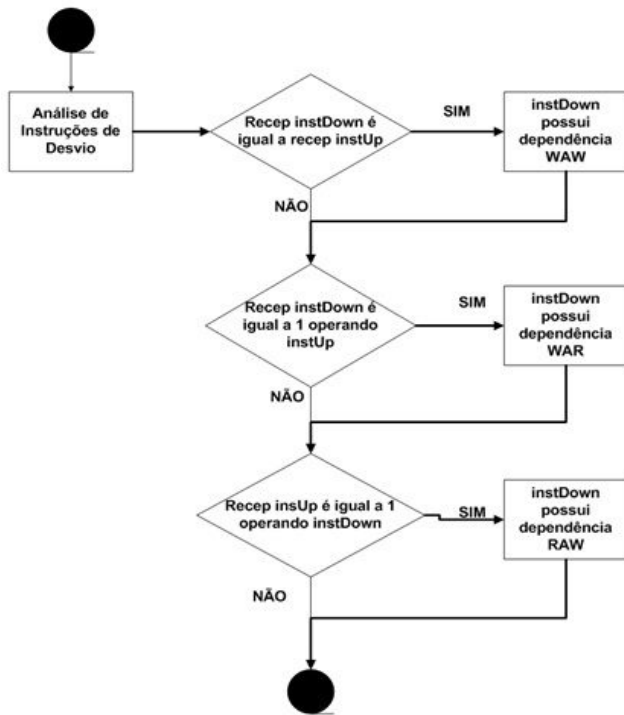


Figura 3. Fluxograma do algoritmo (Fonte: Autor)

Para a análise de dependências é realizada uma estrutura de repetição que percorre o *array* de instruções a partir da segunda instrução até a última. Em cada iteração desta repetição cada instrução é atribuída a uma instância auxiliar de instruções, definida em *instDown*. Em outra repetição interna cada instrução *i-1*, em que *i* representa a posição de *instDown*, é atribuída a *instUp*, onde serão realizadas as comparações, como ilustra a Figura 3.

Cada uma das dependências é verificada da seguinte forma:

- Dependência do tipo WAR: compara-se o registrador destino da instrução *i* com os operandos da instrução *i-1*, caso um deles seja igual, a instrução *i* tem dependência do tipo WAR com a instrução *i-1*.
- Dependências do tipo RAW: compara-se o registrador destino da instrução *i-1* com os operandos da instrução *i*.
- Dependências do tipo WAW: compara-se o registrador destino de *i* com o registrador destino de *i-1*.
- Dependências de desvio: Prioritariamente é verificada a ocorrência de instruções de desvio, para tal, é percorrida a lista até chegar na instrução de desvio “*i*”.

O uso do simulador no contexto da sala de aula pressupõe a interação do aprendiz por meio da inserção no ambiente de instruções na linguagem assembly. Para isso, o aprendiz conta com uma interface visual que permite a entrada de dados contendo dois registradores de operandos juntamente com o número de ciclos que a instrução levará para executar. Ressalta-se que tal característica foi adicionada para fins de simular instruções mais

demoradas. A Figura 4 apresenta a interface para entrada de dados.

```

1  ADD R1,R3 1
2  ADD R2,R1 1
3  DIV R1,R9 1
4  SUB R4,R9 3
5  DIV R5,R6 1
6  MUL R7,R6 1
7  ADD R9,10h 1
8  BNQ R9,10h, ROT 1
9  ADD R7,R9 1
10 ADD R7,1h 1
11 ADD R10,R6 1
12 ROT 1
    
```

Figura 4. Entrada de dados para a simulação. (Fonte: Autor)

Para facilitar o processo de análise as instruções em *assembly* foram mapeadas para a sua forma de execução, facilitando assim o seu entendimento e tratamento no escopo do simulador. A Figura 5 apresenta as instruções presentes na Figura 4 na sua forma de execução.

```

Forma de execução das instruções:

R1 = R1 + R3 -----> com 1 ciclos de execucao
R2 = R2 + R1 -----> com 1 ciclos de execucao
R1 = R1 / R9 -----> com 1 ciclos de execucao
R4 = R4 - R9 -----> com 3 ciclos de execucao
R5 = R5 / R6 -----> com 1 ciclos de execucao
R7 = R7 * R6 -----> com 1 ciclos de execucao
R9 = R9 + 10H -----> com 1 ciclos de execucao
SE R9 = 10H DESVIA PARA ROT -----> com 1 ciclos de execucao
R7 = R7 + R9 -----> com 1 ciclos de execucao
R7 = R7 + 1H -----> com 1 ciclos de execucao
R10 = R10 + R6 -----> com 1 ciclos de execucao
    
```

Figura 5. Forma de execução das instruções. (Fonte: Autor)

A análise da entrada do usuário resulta em três informações de saída: (a) a forma de execução das instruções (representado na Figura 5); (b) as colunas de decodificação, janela, execução e escrita e, (c) o tipo de dependência que cada instrução possui com as demais instruções, possibilitando verificar o conflito que gera uma dependência de dados.

A Figura 6 apresenta a interface prevista exposição dos resultados obtidos após a simulação. Primeiramente são mostradas as instruções que foram decodificadas. Na tabela “Janela” são apresentadas as instruções que estão na janela de instruções, ou seja, as instruções que estão na espera para executar. Na tabela “execução” são mostradas as instruções em execução conforme o número de ciclos que cada uma leva para executar, cada linha representa um ciclo de clock. As colunas UF1, UF2 e UF3 representam as unidades funcionais disponibilizadas para a simulação, de acordo com tipos que podem ser configurados. Por fim é possível visualizar as dependências que as instruções possuem.

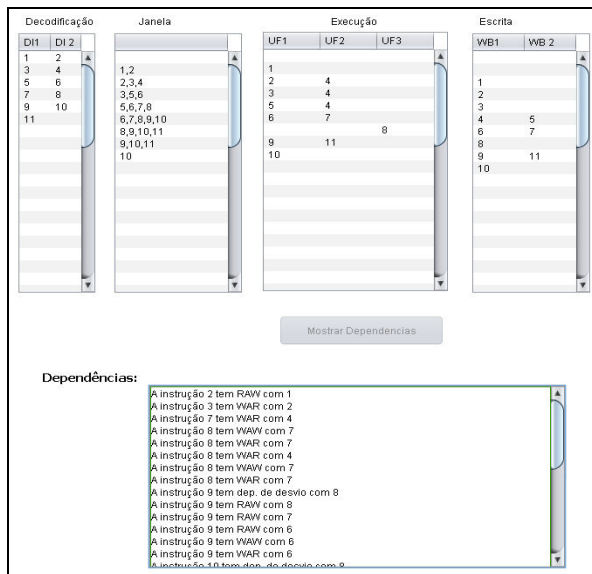


Figura 6. Interface para apresentação dos resultados ao usuário. (Fonte: Autor)

Para fins de verificar a efetividade da utilização do simulador, foram realizados testes em sala de aula, onde foi disponibilizado para um grupo de alunos, 3 (três) atividades a serem resolvidas manualmente e com o auxílio do Simulador AD3W. Para a realização das atividades 1, 2 e 3 foram disponibilizados 10, 20 e 30 minutos, com diversos exercícios contendo 10 instruções cada. O gráfico da Figura 7 elenca os resultados de tal teste mostrando a quantidade de exercícios resolvidos manualmente e com o auxílio do simulador.



Figura 7. Gráfico dos testes realizados. (Fonte: Autor)

Como pode ser verificado houve um aproveitamento considerável em relação ao tempo para a realização dos exercícios o que reflete um melhor entendimento do conteúdo por parte dos alunos. Isso comprova que o uso de ferramentas de auxílio para ensino do tópico de dependência de dados no contexto de arquiteturas paralelas facilita o aprendizado, além de tornar as aulas mais atrativas aos aprendizes.

4. CONSIDERAÇÕES FINAIS

Com a apresentação do simulador AD3W é proposta uma forma alternativa, de maneira simples e rápida, para ensino dos conceitos de paralelismo em nível de instrução com identificação de diferentes tipos de dependências. Pode-se destacar a forma de interação com o aluno, pois, o simulador possibilita a configuração de acordo com as características dos exercícios a realizar (quantidade de unidades funcionais, quantidade de ciclos de clock por classe de instrução).

De acordo com a literatura estudada e os testes realizados, foi possível verificar como os simuladores podem ajudar no processo de ensino de diversos conceitos de arquitetura de computadores. Com base nos testes realizados pode-se verificar um aproveitamento de mais de cem por cento no tempo para a realização efetiva dos exercícios. Outros benefícios identificados pelo uso do simulador foi à redução do tempo de correção e também um melhor entendimento do conteúdo por parte dos alunos.

5. REFERENCIAS

- [1] A. I. T. Ribeiro; A. Rimsa. Técnica Motivacional para o Ensino de Arquitetura de Computadores com Ênfase nos Grandes Desafios da Computação. Workshop sobre Educação em Arquitetura de Computadores - WEAC 2008.
- [2] A. L. Almeida; M. F. O. Araujo. Utilização de Ferramentas Multimídia Para a Construção de Simuladores de Fenômenos Físicos. Universidade do Estado da Bahia. BA. Brasil. 2008.
- [3] A. N. Gonçalves; R. C. L. Silva; R. A. L. Gonçalves; J. A. Martini; R10k: Um Simulador de Arquitetura Superescalar. Workshop Sobre Educação em Arquitetura de Computadores – WEAC, 2007.
- [4] BRORSSON, M. MipsIt: a simulation and development environment using animation for computer architecture education. In Proceedings of 2002 Workshop on Computer Architecture Education: Held in Conjunction with the 29th international Symposium on Computer Architecture (Anchorage, ACM, New York, NY, 12. p. WCAE'02.Alaska),p.1 -8.
- [5] D. A. Patterson, J. L. Hennessy. Arquitetura de Computadores: uma abordagem quantitativa. 4. ed. Rio de Janeiro: Elsevier, 2008.
- [6] DEITEL, H. M. Java, como programar. Porto Alegre: Bookman. 2003.
- [7] GRUNBACHER, H., Teaching computer architecture/organisation using simulators, 28th Annual Frontiers in Education Conference, p.1107-1112 vol. 3, 1998.
- [8] Heckler Valmir. Saraiva M. de F. O. Filho K. de S. O. Uso de Simuladores, Imagens e Animações como Ferramentas Auxiliares no Ensino/Aprendizagem de Óptica. Revista Brasileira de Ensino de Física, v. 29, n. 2, p. 267-273, 2007.
- [9] J. Dongarra,, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White. Sourcebook of Parallel Computing. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2003.
- [10] L. M. Coutinho; J. L. Mendes; C. A. Martins; Web-MHE: Ambiente web de auxílio ao aprendizado de hierarquia de memória. Workshop sobre Educação em Arquitetura de Computadores - WEAC 2006.
- [11] Martins. Carlos Bazílio. Detecção de Paralelismo a partir de Semântica Denotacional e de Grafos de Dependências. Dissertação de Mestrado. Pontifícia Universidade Católica. Rio de Janeiro. Brasil. 2000.