Execução Especulativa

Conceitos Gerais e Técnicas

Daniel Carlos Guimarães Pedronette – RA050269
Instituto de Computação, Universidade de Campinas
Cidade Universitária Zeferino Vaz
Campinas, São Paulo, Brasil
daniel.pedronette@students.ic.unicamp.br

RESUMO
Para que o potencial de desempenho oferecido pelos processadores com *pipeline* seja explorado, é necessário que não haja paralisação do processador por conta de dependências no fluxo de execução.

A execução especulativa é uma técnica que tem como objetivo reduzir impactos de latência, executando instruções antes que suas dependências sejam totalmente resolvidas.

Esse trabalho examina de forma geral os principais conceitos e taxonomia da técnica, relaciona suas principais variações e faz uma descrição mais cuidadosa das técnicas aplicadas às dependências de dados.

Categoria
Arquitetura de Computadores

Termos Gerais
Execução Especulativa

Palavras chave
Pipeline ,Depenências de Controle, Dependência de Dados

1. INTRODUÇÃO

O paralelismo a nível de instrução, possibilitado pelo recurso de *pipeline* nos processadores, representa a execução de diferentes ciclos de instruções distintas, ou mesmo a execução simultânea de múltiplas instruções, distribuídas entre diferentes unidades funcionais. Todavia, pode haver restrições que impeçam instruções distintas de serem realizadas paralelamente ou condicionem o início de execução de uma instrução ao término de outra. A existência ou não dessas dependências determina o grau do paralelismo a nível de instrução, ao qual está condicionado também o desempenho que pode ser alcançado pelo processamento de forma geral.

Pode haver diversos tipos de restrições, ou dependências. Uma das possíveis restrições é a concorrência pela utilização da mesma unidade funcional. Uma arquitetura que dispõe de apenas uma unidade funcional não pode, por exemplo, ter duas operações aritméticas escalonadas ao mesmo tempo. Esse tipo de restrição é usualmente denominada como *functional hazard*. Alguns algoritmos são capazes de realizar um escalonamento adequado para instruções em teoria incompatíveis, minimizando dessa forma os inconvenientes desse tipo de dependência restrição.

Existem também outros tipos de restrições que, em teoria, ditam a ordem pela qual instruções devem ser executadas: as dependências de dados e de controle. Frequentemente essa ordem limitam o paralelismo que pode ser extraído de programas seqüenciais, reduzindo o desempenho. As dependências de dados ocorrem quando uma instrução necessita de um valor que ainda não foi calculado. As dependências de controle, por sua vez ocorrem quando há paralisação do *pipeline*, em virtude da espera pela decisão do fluxo de execução. Ambos os tipos de dependências podem ter seus efeitos ser reduzidos ou eliminados através de análise e execução especulativa.

Execução especulativa consiste em um conjunto de técnicas para antecipar a execução de instruções

antes que todas as dependências tenham sido resolvidas. A não resolução de todas as dependências pode significar inclusive que instruções são executadas desnecessaria ou erroneamente. Todavia, o ganho representado pela não paralisação do *pipeline* é comumente maior que o custo de possíveis execuções equivocadas.

A antecipação proporcionada pela execução especulativa, na maioria dos casos, é baseada em análise estatística dos resultados previamente obtidos. Essa análise é realizada explorando-se a localidade de valor, espaço e tempo, que é observada na imensa maioria dos softwares desenvolvidos. A figura 1 ilustra essa propriedade comum dos softwares como três dimensões, aplicadas em técnicas importantes de execução especulativa.

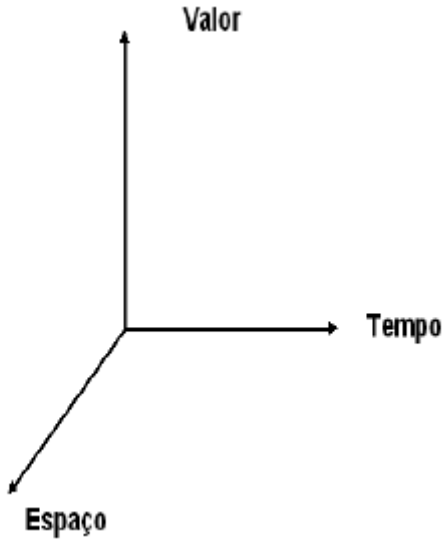

Figura 1. Três dimensões exploradas pela execução especulativa

2. CLASSIFICAÇÃO DAS TÉCNICAS

As técnicas de execução especulativa são comumente agrupadas e classificadas na literatura de acordo com o tipo de dependência que pretende solucionar. São listadas abaixo, e ilustradas na figura 2, as principais técnicas e as dependências relacionadas.

As Dependências de Controle consistem naquelas relacionadas ao fluxo de execução do programa, ou seja, na relação entre duas instruções de maneira que a execução de uma delas determina se a outra deverá ou não ser executada. Pode-se citar algumas técnicas de especulação de controle classificadas da seguinte forma:

- Branch Prediction: Técnicas que tentam predizer o fluxo de execução do programa após uma instrução de desvio com base na probabilidade desse ocorrer.
- Eager Execution: Nessa técnica todos os possíveis caminhos de execução são testados.
- Disjoint Eager Execution: Variação da técnica Eager Execution em que as restrições de disponibilidade de recursos são levadas em consideração para determinação de quais caminhos, selecionados de acordo com a probabilidade de serem tomados, serão executados.

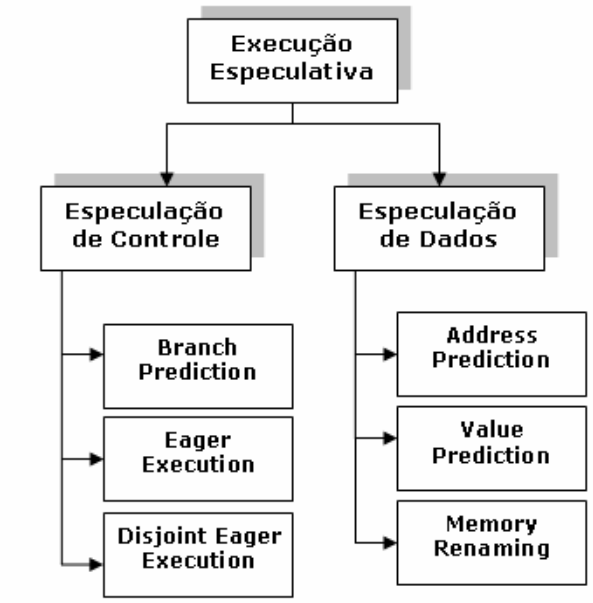

Figura 2. Classificação e taxonomia das técnicas de execução especulativa.

As Dependências de Dados ocorrem quando uma instrução depende de um dado que ainda não foi obtido ou calculado por outra instrução precedente. Principalmente em virtude da latência da instrução *load*, as dependências de dados consistem num importante gargalo para os processadores com

pipeline. As seguintes técnicas de execução especulativa procuram reduzir esse problema:

- Address Prediction: Técnicas que tentam predizer em qual posição de memória dados ou instruções estão armazenados.
- Value Prediction: Técnicas que procuram predizer qual o valor está armazenado em um registrador ou em um endereço de memória.
- Memory Renaming: Técnicas que procuram comunicar valores já armazenados para instruções *loads*.

3. ADDRESS PREDICTION

Todas as instruções *load* necessitam que seu endereço efetivo seja calculado, até que possam ser executadas. Se uma dessas instruções encontra-se no caminho crítico de execução, seria benéfico para o desempenho se o endereço da instrução pudesse ser previsto e assim, o dado alvo carregado tão breve quanto possível.

Dessa forma, as técnicas de predição de endereços procuram basear-se no conceito de localidade temporal, ou seja, posições de memória uma vez acessadas tendem a sê-lo novamente no futuro, para tentar predizer os endereços e aumentar o desempenho na execução de programas. A figura 3, adaptada de Reiman e Carmen [1], procura ilustrar como a predição de endereços reduz o efeitos das dependências.

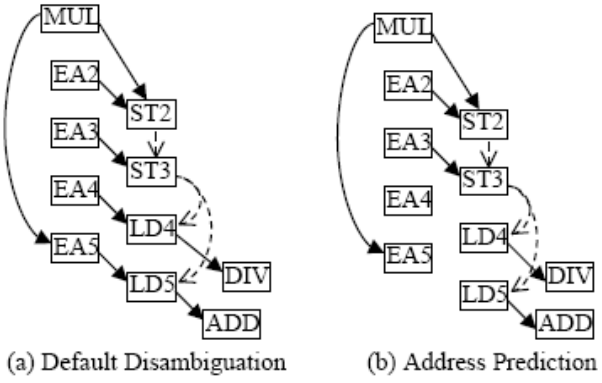

Figura 3. Figura (b) mostra como o LD5 pode ser executado mesmo antes que seu Efective Address tenha sido calculado

Um estudo sobre a possibilidade de predição de instruções *load* é apresentado em Gonzalez e Gonzalez [3] com bons resultados. Em geral, endereços utilizados por mesmas instruções de load ou store seguem uma progressão aritmética ou, em outras palavras, diferem do endereço utilizado na execução anterior apenas por uma constante. Vale salientar que esse fato é bastante típico, principalmente no que diz respeito ao acesso de vetores na memória e ressaltando o conceito de localidade espacial.

Baseado nesse resultado é proposta uma estratégia simples e eficaz para predição de endereços chama da MAP (Memory Address Prediction). Essa estratégia consiste em determinar os endereços durante a decodificação das instruções através de uma tabela chamada Memory History Table (MHT).

Essa tabela é indexada através dos últimos bits da instrução e contém três campos: o endereço anterior, o valor do passo e um contador, cujo bit mais significativo indica se a instrução pode ser predita ou não

Instruções executadas especulativamente devem ser verificadas. Em caso de acertos, o contador da MHT é incrementado e o endereço atualizado. Em caso de erro, o contador é decrementado e endereço e passo são modificados para que possam refletir a nova realidade. O estudo mostra uma avaliação do mecanismo utilizando o SPEC95, mostrando bons ganhos de desempenho.

4. VALUE PREDICTION

Técnicas de predição de valores são baseadas no conceito de localidade de valor, ou seja, a probabilidade de um mesmo valor ser encontrado em sucessivos acessos ao conteúdo de um registrador ou endereço de memória.

No estudo apresentado por Lipasti e Shen [4], são listados motivações que procuram justificar a existência de localidade de valor em programas reais, como por exemplo:

- Redundância de dados, como ocorre em matrizes esparsas.
- Constantes do programa.

• Spill de registradores.

No mesmo estudo, foram realizados realizados uma série de experimentos para mensurar a localidade de valor em leituras da memória e acesso a registradores utilizando um benchmark de 17 programas, utilizando o SPEC95.

Na figura 4 é mostrada graficamente uma avaliação de localidade de valor para instruções load realizada por esses pesquisadores. A localidade apresentada por cada um dos programas listados no eixo horizontal foi estimada contando o número de instruções nas quais o valor lido corresponde a um valor obtido em uma execução anterior da mesma instrução e dividindo pelo total de instruções de leitura.

São apresentadas duas estimativas. Na primeira delas, representada pelas barras claras, é verificado se o valor lido corresponde ao obtido na execução imediatamente anterior.

Na segunda, representada pelas barras escuras, é considerado um histórico de seis execuções.

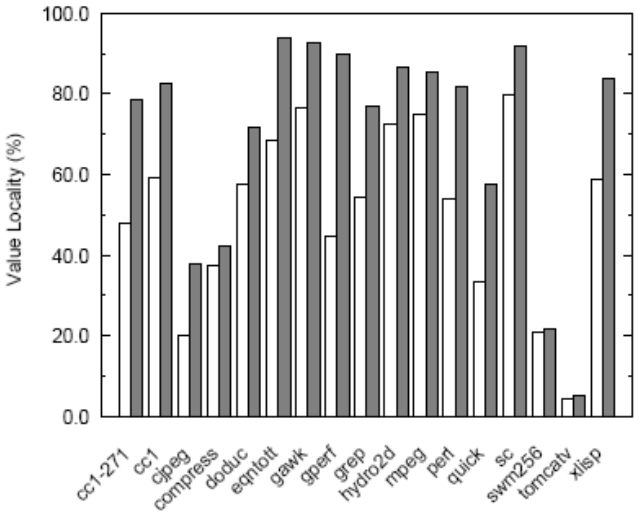

Figura 4: Localidade de valor em acessos à memória

Como pode ser observado, a grande maioria dos programas apresenta correspondência entre 40 e 60% para o primeiro caso e uma significativa melhora desse valor quando um histórico maior é utilizado, alcançando índices superiores a 80%. Apenas três programas (cjpeg, swm e tomcatv) apresentam pouca localidade.

Como pode ser observado, a grande maioria dos programas apresenta correspondência entre 40 e 60% para o primeiro caso e uma significativa melhora desse valor quando um histórico maior é utilizado, alcançando índices superiores a 80%. Apenas três programas (cjpeg, swm e tomcatv) apresentam pouca localidade.

A figura 5 representa a aplicação da mesma metodologia para estimativa de localidade de valor em registradores, ou seja, foi contado o número de vezes em que é escrito em um registrador um valor previamente armazenado nele, dividindo-o pelo número total de escritas em registradores. Quando apenas o histórico mais recente é utilizado, o índice médio de localidade apresentado pela maior parte dos programas avaliados fica em torno de 50%, para um histórico de quatro valores, essa taxa é de 60% aproximadamente.

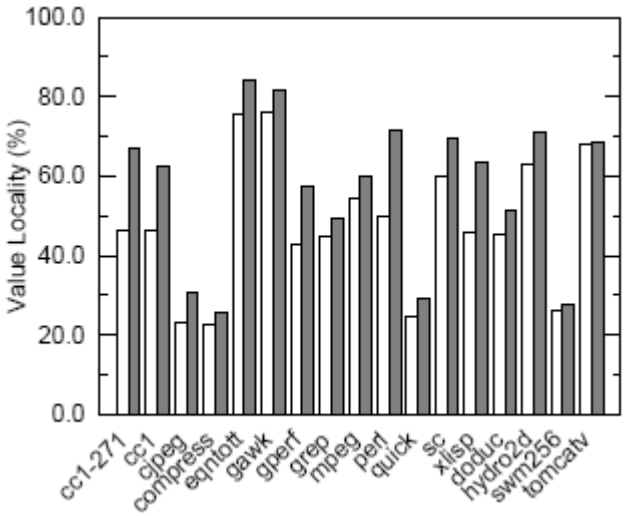

Figura 5. Localidade de valor em escritas a registradores

Para explorar a localidade de valor apresentado pela maioria dos programas, adicionando uma grau maior de liberdade para o escalonamento de instruções, uma melhor utilização dos recursos disponíveis e, possivelmente, uma redução do tempo de execução, o trabalho apresenta uma implementação uma mecanismo de predição de valores baseadas em duas

unidades: uma de predição de valores e outra de verificação.

A unidade de predição de valores é composta por duas tabelas indexadas através dos últimos bits de endereço da respectiva instrução, como mostrado na figura 6, extraída do citado trabalho.

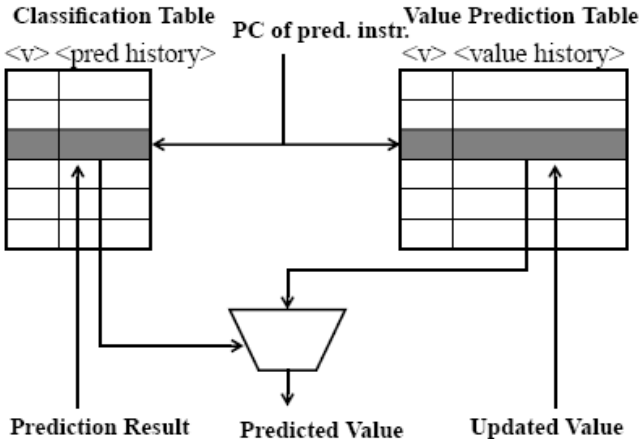

Figure 6: Unidade de Predição de Valores

A tabela de classificação armazena um contador que é incrementado ou decrementado de acordo com acertos ou erros de predição e é utilizado para classificar a instrução como preditível ou não. A tabela de predição contém o valor esperado. Ambas possuem um campo de válido, que pode ser um bit indicando se a entrada é válida ou não ou um campo que deve ser comparado aos bits mais significativos do contador de instruções para verificar a validade da respectiva entrada.

5. MEMORY RENAMING

A técnica de Memory Renaming consiste basicamente no processo de localização das dependências entre instruções *store* e *load*. Uma vez idenficadas essas dependências é possível realizar a previsão, comunicando o dado armazenado pela instrução *store* para a instrução *load* na sequência.

Em Tyson e Austin [5] é apresentada uma técnica para comunicação efetiva de memória, ilustrada na figura 7. A arquitetura é constituída por uma Store Cache (1) para armazenar instruções store recentes (4K entradas diretamente mapeadas), uma Store/Load Cache (2) para aramazentar dependências localizadas (4K entradas diretamente mapeadas) e uma Value File (3) para predição de valores. A técnica conta tamb´me com um mecanismo responsável por determinar quando usar a predição.

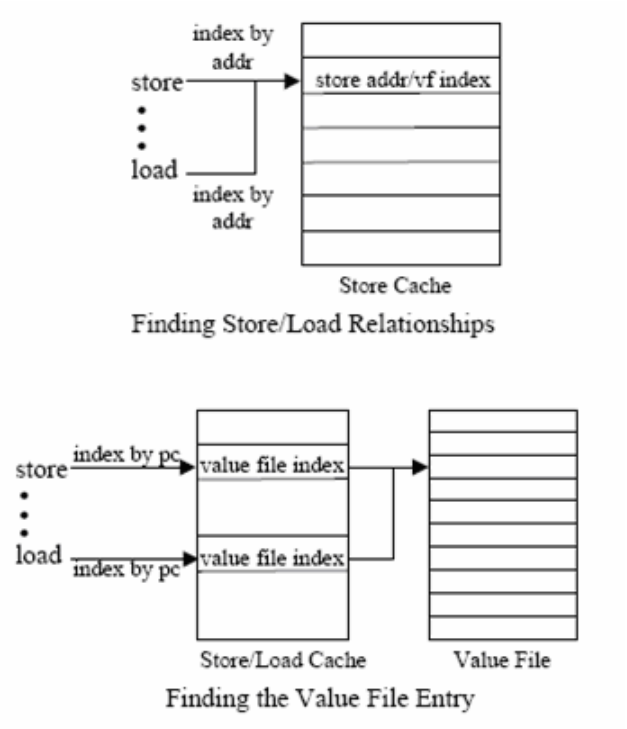

Figura 7. Técnica de predição de valores

Quando uma instrução *store* é decodificada, esta é indexada na Store/Load Cache com o store PC para localizar a entrada Value File. Se não localizada, a instrução *store* é alocada como entrada recente na Value File e é atualizada uma nova entrada na Store/Load Cache.

6. PREDIÇÕES INCORRETAS E EXECEÇÕES

Visando garantir a correta execução do programa as técnicas de execução especulativa devem funcionar de maneira que o resultado final produzido não seja alterado. Para tanto, deve existir mecanismos eficientes de recuperação para predições incorretas, além do devido tratamento de exceções. Ou seja, a ocorrência de uma exceção durante a execução de

código especulativo não pode alterar o estado do programa até que a verificação da predição seja feita.

Como descrito em [1], as predições incorretas podem ser corretamente são tratadas de duas formas:

- Squash Recovery: quando ocorre um predição incorreta, todas as instruções no reorder buffer são invalidadas
- Reexecução: realiza novamente a execução das instruções diretamente ou indiretamente dependentes da predição incorreta.

Há diversas soluções para o problema do tratamento de exceções durante execuções especulativas. A abordagem mais simples, como descrito em [2], consiste em não especular em instruções possíveis de gerarem exceções, conhecido como modelo de especulação restritivo.

7. CONCLUSÕES

Apresentamos de forma geral as principais restrições de desempenho nos processadores com pipeline, e como as técnicas de Execução Especulativa podem auxiliar a reduzir o efeito dessas limitações. Uma classificação e taxonomia das técnicas foram apresentadas, em relação ao tipo de dependência, de controle ou de dados, que esta procura solucionar. Após essa conceituação geral, apresentamos ao longo do texto algumas abordagens de execução especulativa de dados, assim como algumas respectivas análises de desempenho. Por fim citou-se a importância do tratamento exceções e de predições incorretas.

Ao longo do trabalho podemos observar a relevância da aplicação da técnica, propiciando o aumento do grau de paralelismo a nível de instrução e evitando paralisações do processador. Todavia, o aumento de desempenho obtido é dependente da técnica utilizada e das características do programa em execução. Isso significa que técnicas de execução especulativa, especialmente as que realizam predições, terão efeitos mais significativos em programas que apresentam alta localidade.

Um trabalho futuro e mais extensor seria uma análise comparativa entre melhorias em desempenho obtida com aplicação de diferentes técnicas, avaliadas individualmente e através de possíveis combinações.

8. REFERÊNCIAS

[1] B. Calder and G. Reinman, "A Comparative Survey of Load Speculation Architectures," Journal of Instruction-Level Parallelism 1, 2000.

[2] A. R. Ganesh Lakshminarayana and N. K. Jha. Incorporating speculative execution into scheduling of control-flow-intensive designs. IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, pages 308–324, March 2000.

[3] J. Gonzalez and A. Gonzalez. Memory address prediction for data speculation. Technical report, Universitat Politecnica de Catalunya, 1996.

[4] M. H. Lipasti and J. P. Shen. Exploiting value locality to exceed dataflow limit. International Journal of Parallel Pogramming, 26(4):505–538, 1998.

[5] G. Tyson and T. M. Austin. Improving the accuracy and performance of memory communication through renaming. In 30th Annual International Symposium on Microarchitecture, pages 218–227, December 1997.