Execução Especulativa

Conceitos, Princípios e Técnicas

Patrícia Pilisson Côgo
RA 041481
Instituto de Computação, Universidade de Campinas
Campinas, São Paulo, Brasil
patricia.cogo@ic.unicamp.br

RESUMO

Execução especulativa, também denominada avaliação especulativa, é uma das mais eficientes técnicas para aumento de desempenho em processadores com *pipeline*. Essa técnica consiste em executar instruções antes que suas dependências, sejam elas de dados ou de controle, tenham sido completamente resolvidas.

Técnicas de execução especulativa exploram as localidades de espaço, tempo e valor apresentadas pela maior parte dos softwares desenvolvidos. Dessa maneira, dependências relativas às instruções de tomada de decisão e determinação do valor dos operandos, que impõem limites à paralização a nível de instrução, podem ser contornadas, aumentando o número de instruções disponíveis para serem escalonadas e, por conseqüência, o grau de paralelismo obtido para o programa em execução.

Termos Gerais

Execução Especulativa

Palavras Chaves

Dependências de controle, dependências de dados, execução especulativa

1. INTRODUÇÃO

Processadores com Pipeline permitem a execução simultânea de múltiplas instruções distribuindo-as entre suas diferentes unidades funcionais. Todavia, podem existir restrições que impeçam duas instruções de serem realizadas paralelamente ou condicionem o início de execução de uma instrução ao término de outra. O grau de indepêndencia entre instruções de um mesmo programa é chamado *Paralelismo a Nível de Instrução*, e quanto maior for esse paralelismo, maior o desempenho alcançado durante seu processamento.

Uma das restrições é a concorrência pela utilização da mesma unidade funcional. Uma arquitetura que dispõe de apenas uma ULA[1] não pode, por exemplo, ter duas operações aritméticas escalonadas ao mesmo tempo. Usualmente denominada na literatura como *Functional Hazard*, existem algoritmos de escalonamento capazes de minimizar os efeitos dessa restrição, como o implementado no compilador GCC [5].

Outras restrições que ditam a ordem pela qual instruções devem ser executadas e limitam o paralelismo que pode ser extraído de programas seqüênciais são as dependências de dados e de controle, cujos efeitos podem 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. Esse antecipação, na maioria dos casos, é baseada em análise estatística dos resultados previamente obtidos, explorando a localidade de valor, espaço e tempo apresentada pela imensa maioria dos softwares desenvolvidos, graficamente representadas na figura 1.

Dependência de controle é relação entre duas instruções de maneira que a execução de uma delas determina se a outra será executada ou não. Técnicas de especulação de controle podem ser 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 anterior 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.

Dependência de dados é relação entre instruções na qual a execução de uma é dependente do resultado gerado pela outra. Técnicas de especulação de dados podem ser de duas naturezas:

[1]Unidade Lógica e Aritmética

- **Predição de endereços**: Técnicas que tentam predizer em qual posição de memória dados ou instruções estão armazenados.
- **Predição de valores**: Técnicas que procuram predizer qual o valor está armazenado em um registrador ou em um endereço de memória.

A figura 2 resume graficamente a taxonomia das técnicas de execução especulativa.

Os resultados gerados por operações executadas especulativamente são chamados *Resultados Especulativos*, e podem ser utilizados ou descartados dependendo da avaliação de suas dependências nos ciclos de execução posteriores. Quando uma instrução é executada todos os resultados especulativos relativos a ela devem ser resolvidos [2].

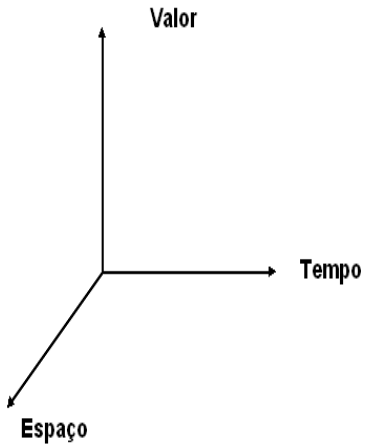

Figure 1: Três dimensões de localidade encontradas em softwares

Nas próximas seções são descritas em maiores detalhes e exemplificadas cada uma dessas técnicas: na seção 2 são tratadas especulações de controle e na seção 3 especulações de dados. Na seção 4 é discutido o tratamento dado a exceções quando há suporte à execução especulativa e na seção 5 são apresentadas as conclusões finais.

2. ESPECULAÇÃO DE CONTROLE

Como visto na seção 1, técnicas de especulação de controle podem ser classificadas como *Branch Prediction*, *Eager Execution* ou *Disjoint Eager Execution*. A seguir serão descritas cada uma delas.

2.1 Branch Prediction

Especulações de controle baseadas na aplicação de técnicas de *Branch Prediction* consistem em determinar qual será o caminho de execução de um programa a partir de uma instrução de desvio, antes que essa tenha sido completamente resolvida. A adoção de um caminho é feita com base na sua probabilidade de ser tomado, a qual pode ser determinada *a priori*, através da aplicação de *benchmarks*, ou dinamicamente, durante a execução do programa. Há dois tipos de especulações de controle: as que procuram predizer direção e as que procuram predizer o endereço alvo do desvio.

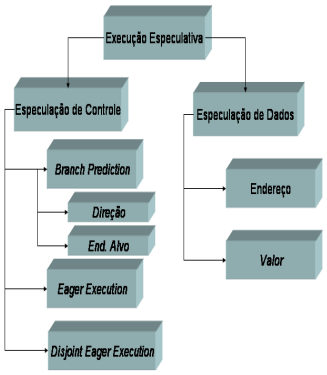

Figure 2: Classificação das técnicas de execução especulativa

A primeira delas consiste numa decisão binária que indicará se o desvio será tomado ou não. O aumento de desempenho promovido por essa técnica está na manutenção da localidade espacial do programa, possibilitando que instruções continuem sendo buscadas e colocadas no *pipeline* de execução. Naturalmente, uma predição errada consiste em um atraso ainda maior que a execução não especulativa, pois o estado do processador anterior à execução especulativa deverá ser restaurado e as instruções pertencentes ao caminho correto executadas. Entretanto, o ganho em desempenho obtido através das técnicas de predição mantém a execução especulativa mais eficiente [6].

Esse método é particularmente eficaz em estruturas de repetição, como os comandos *for*, *repeat* e *while*, onde a tomada de desvio ocorre na grande maioria dos casos, resultando em elevadas taxas de acerto. Smith descreve um conjunto de estratégias para implementação de técnicas dessa natureza [6].

A segunda forma de especulação consiste numa decisão multivalorada que procura predizer que ponto do espaço de endereçamento do programa será alvo do desvio. Técnicas dessa natureza estão implementadas na arquitetura na maioria dos processadores superescalares modernos, como PowerPC 604/620 e Pentium Pro, que precisam de alguma forma realizar busca especulativa de instruções para colocá-las em execução [4].

Essas arquiteturas necessitam implementar um mecanismo rápido e acurado de busca de endereços de instruções, a fim de diminuir a probabilidade e o custo de especulações incorretas. Yeh e Patt sugerem um mecanismo para busca de instruções em processadores com suporte a execução especulativa baseado em um preditor adaptativo e um buffer de armazenamento de endereços alvos de desvio [8].

Muito atrativas em relação ao custo de execução, que cresce linearmente em relação ao número de desvios que sofreram especulação, essas técnicas apresentam alta acurácia. Os melhores métodos conseguem taxas de acerto na predição de desvios entre 90% e 96% [7].

2.2 Eager Execution

Uma alternativa à técnica de *Branch Prediction*, consiste em executar todos os possíveis caminhos concorrentemente. O programa é representado como uma árvore, em que cada nó representa uma instrução de desvio e as arestas são blocos básicos. Todos os blocos em um mesmo nível são executados paralelamente.

Apesar da simplicidade, alto desempenho e da garantia de execução correta, esse esquema é baseado na situação hipotética em que não há restrições quanto ao uso de recursos de execução. O custo de execução é muito alto, pois o número de unidades de processamento necessárias cresce exponencialmente em relação ao número de desvios existentes, o que torna essa estratégia impráticável.

2.3 Disjoint Eager Execution

Disjoint Eager Execution é uma técnica que combina o baixo custo da implementação das técnicas de *Branch Prediction* e o elevado desempenho da técnica de *Eager Execution*, apresentando desempenho melhor que ambos quando há restrições quanto aos recursos de execução disponíveis [7].

Essa técnica consiste em alocar unidades de execução para blocos com maior *probabilidade acumulada* (*pa*), definida como o produto entre a probabilidade do bloco ser executado e a probabilidade de todos os blocos sucessores.

Quanto a probabilidade de um caminho ser tomado ou de não ser tomado é próxima a 100%, para todos os desvios, *Disjoint Eager Execution* se aproxima da técnica de *Branch Prediction*. Quando essa probabilidade se aproxima de 50%, essa técnica se torna equivalente a técnica de *Eager Execution*.

Essas probabilidades devem ser determinadas dinamicamente, sendo re-computadas a cada vez que uma instrução de desvio é executada. Essas multiplições somadas a da escolha da maior *pa* a cada ciclo resultam em custos de execução muito altos, tornando essa abordagem pouco atrativa em termos práticos. Entretanto, Uht e Sintage [7] propuseram uma heurística baseada em probabilidades fixas para cada desvio, estimadas a partir da aplicação de *benchmarks*, que torna a determinação da árvore de execução simples e mantém elevados os índices de desempenho alcançados.

A figura 3, extraída do trabalho de Uht e Sintage [7], é uma comparação da janela de execução entre as diferentes estratégias em uma máquina com seis unidades de processamento.

Nesse exemplo, a acurácia na predição é 0.7. Flexas com orientação à esquerda indicam caminhos preditos e em negrito indicam os blocos que estão em execução. Números circulados indicam o índice da unidade de execução alocada para o bloco que acompanha, enquanto os demais representam as probabilidades acumuladas.

3. ESPECULAÇÃO DE DADOS

Técnicas de especulação de dados podem ser de duas naturezas: as que especulam sobre o endereço de armazenamento e as que especulam sobre o valor armazenado em registradores ou em memória. A seguir serão analisadas cada uma delas.

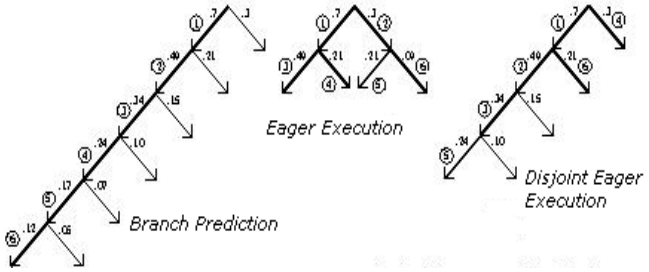

Figure 3: Comparação entre métodos de especulação

3.1 Predição de Endereços

Técnicas de predição de endereços se baseiam no conceito de *localidade temporal*, ou seja, posições de memória uma vez acessadas tendem a sê-lo novamente no futuro, para aumentar o desempenho na execução de programas.

Gonzalez e Gonzalez [3] realizaram um estudo sobre a preditibilidade de instruções de acesso à memória com resultados entusiasmadores. 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.

Com base nesse resultado, os dois pesquisadores propuseram 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, como está representado na figura 4, adaptada do mesmo trabalho.

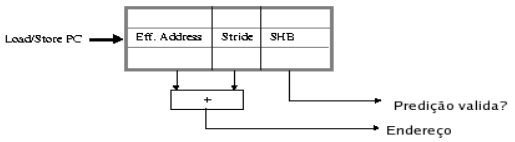

Figure 4: Representação da *Memory History Table*

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 endere co e passo são modificados para que possam refletir a nova realidade.

3.2 Predição de Valores

Técnicas de predição de valores são baseadas no conceito de *localidade de valor*, i.e, probabilidade de um mesmo valor ser encontrado em sucessivos acessos ao conteúdo de um registrador ou endere co de memória.

Lipasti e Shen [4] listam uma série de argumentos que procuram justificar a existência de localidade de valor em programas reais, como por exemplo:

- Redundância de dados, como ocorre em matrizes esparsas.
- Contantes do programa.
- *Spill* de registradores.

No mesmo estudo, Lipasti e Shen realizaram uma série de experimentos para mensurar a localidade de valor em leituras da memória e acesso a registradores utilizando um *benchmark* de 17 programas.

Na figura 5 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.

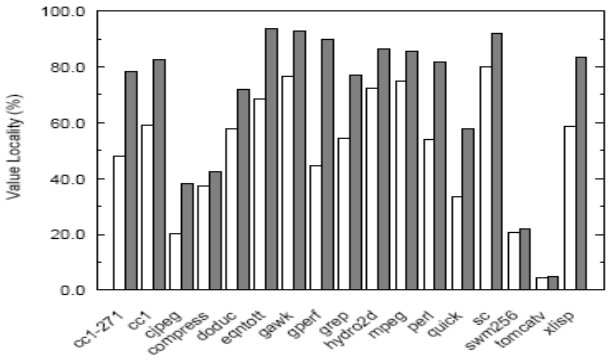

Figure 5: Localidade de valor em acessos à memória

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.

A figura 6 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.

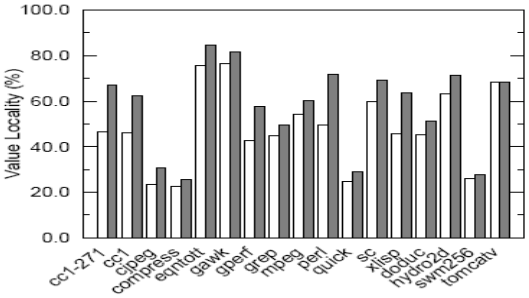

Figure 6: Localidade de valor em escritas a registradores

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, os dois pesquisadores implementaram uma mecanismo de predição de valores basedas 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 7, extraída do citado trabalho. 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.

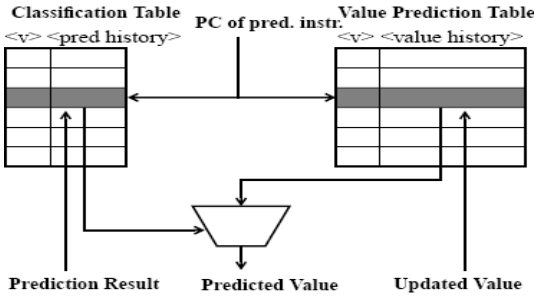

Figure 7: Unidade de Predição de Valores

O funcionamento da unidade de verificação e correção de predições incorretas está representado na figura 8, extraída do mesmo trabalho. No exemplo mostrado são executadas em parelelo duas instruções entre as quais há dependência de dados. A instrução predecessora, à esquerda, tem seu valor predito durante os dois primeiros ciclos e continua sua

execução. A instrução mostrada à direita lê o valor especulado durante seu ciclo de execução e também prossegue normalmente, a única restrição é que a unidade de execução deverá permanecer reservada até que o resultado especulado seja verificado. Em caso de acerto na predição, as unidades são liberadas. Caso contrário, a instrução à direita deve ser re-executada.

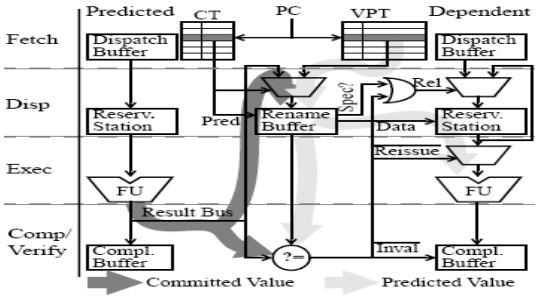

Figure 8: Funcionamento da unidade de verificação e correção

4. TRATAMENTO DE EXCEÇÕES

Todas as técnicas de execução especulativa devem garantir a funcionalidade do programa, de maneira que o resultado final produzido não seja alterado por elas. Para tanto, além de mecanismos de recuperação para predições incorretas, especial atenção deve ser destinada ao tratamento de exceções.

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. Em especial, um programa não pode simplesmente terminar pela ocorrência de uma exceção fatal durante a execução especulativa.

Há diferentes soluções para esse problema. A abordagem mais simples consiste em não especular em instruções possíveis de gerarem exceções, conhecido como *modelo de especulação restritivo* [2].

O chamado *modelo geral de especulação* procura restringir as exceções possíveis de ocorrerem durante a execução especulativa. Em geral, esse modelo permite apenas que exceções de falhas de *cache* e TLB[2], indispensáveis para a continuidade de execução, ocorram. Simples e rápido para ser implementado esse método alcança bom desempenho e requer poucas modificações na arquitetura. Infelizmente, inibe também exceções que poderiam ocorrer na execução não especulativa do programa [1].

Bringmann *et al.* [1] propuseram uma solução chamada *write-back supression*. Nesse modelo o resultado de instruções especulativas é armazenado em um banco de registrados auxiliar. Quando a predição é correta, um mecanismo verifica se houve exceções durante a execução do código especulativo. Se não ocorreram, o banco auxiliar é copiado para o banco de registradores principal. Caso a predição tenha sido incorreta, os valores armazenados nos registros auxiliares são descartados e o código original é executado. Apesar de tratar o problema encontrado no modelo anterior, essa solução representa um aumento significativo no custo de execução.

[2] *Translation-lookaside buffer*

5. CONCLUSÕES

Nesse trabalho foram apresentados conceitos relativos à execução especulativa de instruções, uma otimização capaz de aumentar o paralelismo a nível de instrução. Entre as limitações impostas a esse paralelismo estão a dependência de dados e de controle, que podem ter seus efeitos reduzidos através de técnicas de especulação, algumas delas discutidas ao longo desse texto.

Quaisquer métodos especulativos devem preservar a funcionalidade do programa, de maneira que a realização de otimizações dessa natureza não altere seu resultado final. Para isso, além de métodos de recuperação de predições incorretas, devem ser implementados mecanismos para tratamento de exceções ocorridas durante a execução especulativa, como os apresentados na seção 4.

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 estudo interessante seria uma análise comparativa entre a melhoria em desempenho obtida com aplicação de diferentes técnicas, avaliadas individualmente e através de possíveis combinações. Infelizmente nenhum tópico sobre esse assunto foi encontrado na bibliografia pesquisada.

6. AGRADECIMENTOS

Gostaria de agradecer à FAPESP (processo 05/53279-6) pelo fomento a este trabalho e a todos os outros.

Agradeço também ao criador do Google Scholar, Anurag Acharya, pela ajuda prestada por sua ferramenta ao longo da realização deste.

7. REFERENCIAS

[1] R. A. Bringmann, S. A. Mahlke, R. E. Hank, J. C. Gyllenhaal, and W. mei W. Hwu. Speculative execution exception recovery using write-back suppression. *Proceedings of the 26th Annual International Symposium on Microarchitecture (MICRO-26)*, pages 214–223, 1993.

[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] V. N. Marakov. The finite state automaton based pipeline hazard recognizer and instruction scheduler in gcc. *Proceedings of the GCC Developers Summit*, 2003.

[6] J. E. Smith. A study of branch prediction techniques. *Proceedings of the 8th Annual Symposium of Computer Architecture*, pages 135–147, 1981.

[7] A. K. Uht and V. Sindagi. Disjoint eager execution: An optimal form of speculative execution. *Proceedings of the 28th International Symposium on Microarchitecture, IEEE/ACM*, pages 129–139, November/December 1995.

[8] T.-Y. Yeh and Y. N. Patt. A comprehensive instruction fetch mechanism for a processor supporting speculative execution. *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pages 129–139, 1992.