Trace Cache

Mário Luiz Rodrigues Oliveira - RA 066254
Instituto de Computação/UNICAMP
Av. Albert Einstein, 1251
Campinas/SP, Brasil
mario.oliveira@students.ic.unicamp.br

RESUMO

Processadores superescalares demandam um aumento no *bandwidth* do mecanismo de busca de instruções para melhorar o desempenho. Uma cache de instruções convencional não é capaz de suprir esses processadores com um grande número de instruções por ciclo, pois na maioria das vezes não dispõe de quantidades razoáveis de instruções em um mesmo bloco básico. Uma solução para tal problema é adicionar à cache de instruções convencional uma *trace cache*, a qual captura uma seqüência dinâmica de instruções e as armazena na *trace cache*. Essa técnica foi proposta, em 1996, por *Eric Rotenberg*, o qual mostrou que ela melhora o desempenho em 28%, em média, quando comparada à mecanismos de busca seqüenciais. Para auferir esse desempenho utilizou-se *benchmarks* com inteiros. O presente artigo mostra a proposta inicial de Robenterg, o uso de *trace cache* no processador Pentium 4 e algumas idéias para melhorar o desempenho da técnica de *trace cache*.

Termos gerais

Cache, Desempenho

Palavras-chave

Trace cache, memória cache, desempenho

1. INTRODUÇÃO

A organização de computadores de superescalares é dividida em dois mecanismos: o mecanismo de busca de instruções e o mecanismo de execução de instruções. Entre esses dois mecanismos pode-se colocar *buffers*, tais como filas ou estações de reservas. Conceitualmente, o mecanismo de busca atua como um produtor, o qual busca, decodifica e coloca as instruções no *buffer* e o mecanismo de execução atua como um consumidor, o qual remove as instruções do *buffer* e as executa, de acordo com a disponibilidade de recursos e dados.

Para melhorar o desempenho de processadores superescalares deve-se empregar técnicas agressivas que explorem paralelismo no nível de instrução (ILP) afim de executar várias instruções no mesmo ciclo. Entretanto para tirar proveito do ILP, deve-se ter o máximo possível de instruções no *buffer* existente entre os mecanismos de busca e execução.

Não haveria problemas se esse *buffer* fosse preenchido com instruções de um bloco básico, uma vez que todas as instruções do bloco básico podem ser executadas em paralelo. Um bloco básico é definido como um trecho do programa que não apresenta instruções de desvio, a não ser eventualmente a última instrução e assim, inexistindo conflitos estruturais e de dados, um bloco básico pode ser executado em paralelo aumento o desempenho. Todavia, a quantidade de instruções em um bloco básico é pequena. A tabela 1, adaptada de [7], mostra que o bloco básico, para programa de inteiros, possue em média 4 ou 5 instruções e dessa forma o uso de ILP dentro de apenas um bloco básico é limitado. Para melhorar o desempenho deve-se prover ao mecanismo de execução mais de um bloco básico por ciclo. A introdução de múltiplos preditores de *branches* por ciclo permite alimentar o *buffer* com múltiplos blocos básicos por ciclo e dessa forma consegue-se uma melhoria no desempenho do processador.

Benchmark	**Branch Tomados %**	Tam. médio do bloco	Número de instruções entre branch tomados
Eqntoot	86.2 %	4.20	4.87
Espresso	63.8 %	4.24	6.65
Xlisp	64.7 %	4.34	6.70
Gcc	67.7 %	4.65	6.88
Sc	70.2 %	4.71	6.71
Compress	60.9 %	5.39	8.85

Table 1: Estatísticas de branch e blocos básicos

O restante desse texto está organizado em 6 seções. Na próxima seção explica-se o que é e como funciona a técnica *trace cache*. Na seção 3 mostra-se o uso de *trace cache* no Pentium 4. Na seção 4 apresenta-se a melhoria de desempenho obtida com a adoção da técnica *trace cache*. A conclusão é apresentada na seção 5 e as referências na seção 6.

2. TRACE CACHE

O trabalho da unidade de busca é alimentar a unidade de decodificação. Entretanto as instruções são colocadas na

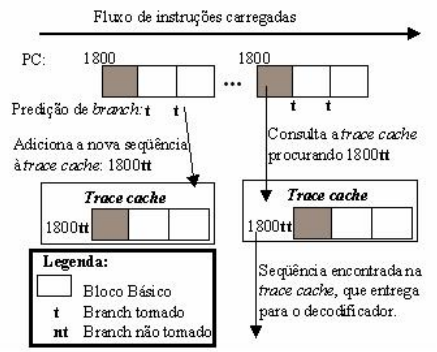

Figure 1: Visão geral do comportamento de uma *trace chace*

cache de instruções na ordem de compilação, o que é favorável a códigos que não tenham *branchs* tomados ou que possuem blocos básicos compostos por muitas instruções. Porém, como visto na tabela 1, esse não é o caso de programas de inteiros.

Uma forma de aumentar a quantidade de instruções disponíveis ao decodificador é através de *trace cache*, técnica proposta por [7] em 1996. A *trace cache* é uma cache adicional à cache de instruções que captura a seqüência dinâmica de instruções de um programa. Uma *trace* é uma seqüência de no máximo **n** instruções e no máximo **m** blocos básicos, iniciando em qualquer ponto da seqüência dinâmica de instruções. O limite **n** é o tamanho da linha da *trace cache* e **m** é o *throughput* do preditor de *branches*. Uma *trace* é completamente especificada através de um endereço inicial dado pelo *Program Counter - PC* e de uma seqüência de saídas dos **m - 1** preditores de *branches*, os quais indicam o caminho a ser seguido, ou seja, se o *branch* foi tomado ou não tomado. A figura 1, retirada de [6], mostra uma visão geral do comportamento de uma *trace cache*.

A *trace cache* é preenchida conforme a execução do programa, a cada *trace* diferente encontrada uma nova linha é alacoda na *trace cahce*. É interessante notar que para um mesmo PC pode-se ter mais de uma linha alocada se a seqüência de saída do preditor de *branch* for diferente. Algumas implementações transformam isso em duas linhas na *trace cache*com o mesmo PC, porém com *bits* diferentes para indicar o novo caminho a ser seguido. Outras implementações permitem apenas uma linha com um determinado PC, nesse caso a *trace* antiga é substituída pela nova.

Ao encontrar o mesmo *trace* novamente, a *trace cache* entrega ao decodificador uma linha, ou seja, todas as instruções constantes em uma *trace*. Caso contrário as instruções são buscadas na cache de instruções convencional.

Pode-se perceber pelo mecanismo de funcionamento da *trace cache* que um bom preditor de *branch* é necessário para se obter bom desempenho, pois a cada predição errada são perdidos vários ciclos até que a instrução seja carregada da memória para cache e dessa para o decodificador do processador.

A *trace cache* apresenta, assim como outros tipos de caches, tamanho limitado e dessa forma deve-se estabelecer algum critério de substituição para as linhas da *trace cache*. Assim, em sua implementação, são usados *bits* adicionais em cada linha e um desses *bits* é o *bit* de validade indicando quais linhas são válidas e um algoritmo de substituição, como por exemplo, o algoritmo LRU [6].

Uma desvantagem no uso de *trace cache* é o fato dela armazenar as mesmas instruções várias vezes na cache de instruções, pois *branches* que fazem escolhas diferentes resultam na inclusão das mesmas instruções como partes de *traces* diferentes, cada um deles ocupando espaço na *trace cache*[3].

3. TRACE CACHE NO PENTIUM 4

Algumas arquiteturas substituiram a cache de instruções convencional por uma *trace cache*, como é o caso dos processadores da *Intel* baseados na micro-arquitetura *NetBurst*, dentre eles o Pentium 4. Esses processadores executam micro-operações que são derivadas do conjunto de instruções IA-32 e por sua característica superescalar executam até três instruções em um ciclo de clock [3] [4].

O uso de *trace cache* no Pentium 4 apresenta vantagens, tais como:

- cada linha da *trace cache* é preenchida com um conjunto de instruções pertencentes a mais de um bloco básico;
- as instruções são armazenadas decodificadas.

A cache nível 1 do Pentium 4 é uma *trace cache* e entrega três micro-operações por ciclo de clock para o mecanismo de execução. Ela é capaz de armazenar até 12.000 micro-operações decodificadas e possui uma taxa de acerto similar à uma cache de instruções convencional com tamanho entre 8 Kbytes e 16 Kbytes [4].

4. MELHORIAS NO DESEMPENHO

O alto desempenho em processadores superescalares é obtido através da execução de várias instruções em paralelo. Para tanto duplicam-se várias unidades funcionais e quanto mais dessas unidades funcionais forem usadas ao mesmo tempo, melhor o desempenho conseguido.

Como visto na Seção 2, *trace cache* é uma proposta para aumentar a quantidade de instruções entregues ao mecanismo

de execução do processador em cada ciclo. Em [7], o ganho médio obtido com o uso de *trace cache* foi de 28% em *benchmarks* com inteiros em comparação com alguns mecanismos convencionais de busca, a saber:

- buscar um bloco básico por ciclo de clock;
- buscar três blocos básicos por ciclo de clock;
- *Branch Address Cache*, maiores detalhes sobre essa técnica podem ser encontrados em [8];
- *Collapsing Buffer*, detalhes dessa técnica podem ser obtidos em [2].

Para auferir essa porcentagem de ganho utilizou-se os *benchmarks* de inteiros SPEC92 e IBS. Robenterg notou, também, que o desempenho da *trace cache* pode ser melhorado aumentando o seu tamanho e/ou a sua associatividade.

Outros autores trabalhando com a técnica de *trace cache* apresentaram novas idéias para melhorar o seu desempenho, alguns trabalhos nesse sentido são:

- *trace cache* seletiva;
- *trace cache* baseada em blocos.

A *trace cache* seletiva é descrita em [5] e sua ideía principal é construir apenas os *traces* com maior freqüência, ou seja, os *traces* que apresentam a maior porcentagem de acertos.

Em [1] é apresentada a *trace cache* baseada em blocos, a qual armazena ponteiros para os blocos que constituem a *trace* ao invés de guardar as próprias instruções da *trace* explicitamente. Tais ponteiros são armazenados em uma tabela de *trace*.

5. CONCLUSÃO

A *trace cache* foi proposta como uma forma de aumentar o *bandwidth* no mecanismo de busca dos processadores superescalares, os quais necessitam de um grande número de instruções decodificadas por ciclo, a fim de apresentarem um ganho significativo de desempenho. E nesse sentido, a técnica de *trace cache* apresentou-se como uma boa solução para o problema.

Cumpre salientar que o desempenho desse tipo de cache está associado a um bom preditor de *branch*, pois quanto melhor os resultados dos preditores de *branches*, melhores os resultados obtidos pela *trace cache*. Dado que atualmente a quantidade de trabalhos na área de predição de *branches* é grande e apresenta resultados muito bons, tal fator não torna-se um empecilho à aplicação da técnica de *trace cache*.

Por fim, nota-se a utlização dessa técnica em novos processadores, como é o caso do Pentium 4 da *Intel*.

6. REFERÊNCIAS

[1] Bryan Black, Bohuslav Rychlik, and John Paul Shen. The block-based trace cache. *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 196 – 207, Maio 1999.

[2] Thomas M. Conte, Kishore N. Menezes, Patrick M. Mills, and Burzin A. Patel. Optimization of instruction fetch mechanisms for high issue rates. *Proceedings of the 22th Annual International Symposium on Computer Architecture*, pages 333 – 344, Junho 1995.

[3] John L. Hennessy and David A. Patterson. *Arquitetura de Computadores: Uma Abordagem Quantitativa*. Editora Campus, São Paulo, 2003.

[4] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel. The microarchitecture of the pentium 4 processor, Junho 2006.

[5] Jie S. Hu, Mary Jane Irwin, N. Vijaykrishnan, and Mahmut Kandemir. Selective trace cache: A low power and high performance fetch mechanism, Junho 2006.

[6] Danilo Lacerda. Trace caches: alternativa inteligente à cache de instruções, Junho 2006.

[7] Eric Rotenberg, Steve Bennett, and James E. Smith. Trace cache: A low latency approach to high bandwidth instruction fetching. *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 24 – 35, Dezembro 1996.

[8] Tse-Yu Yeh, Deborah T. Marr, and Yale N. Patt. Increasing the instruction fetch rate via multiple branch prediction and a branch address cache. *Proceedings of the 7th International Conference on Supercomputing*, pages 67 – 76, Julho 1993.