Trace Caches alternativa inteligente à cache de instruções

Danilo Lacerda
RA: 008448
Instituto de Computação UNICAMP
Av. Albert Einstein, 1251
Campinas/SP Brasil
+55 (19) 9121-0401
danilolacerda@gmail.com

RESUMO

Uma *cache* de instruções convencional não é capaz de entregar ao processador um grande número de instruções a cada ciclo de *clock*, pois muitas vezes não temos essa quantidade de instruções em seqüência no mesmo bloco básico. Para otimizar a taxa de entrega de instruções ao processador complementamos a *cache* convencional com uma *trace cache*. Guardando um histórico das instruções executadas e reaproveitando seqüências de instruções que se repetem, mesmo que separadas por desvios. É possível aumentar o desempenho médio, para *benchmarks* com inteiros, em 67,9% usando a *trace cache*, em comparação com outros mecanismos de *fetch* de instruções e em 16,3% em *benchmarks* de ponto-flutuante [5]. Já outras arquiteturas preferem substituir a *cache* de instruções convencional, pela *trace cache*. Essa é a estratégia da Intel nos seus processadores com a micro-arquitetura *Netburst*, dentre eles o Pentium 4.

Categoria

B.3.2 [**Memory Structures**]: Design Styles – *cache memories.*

Termos gerais

Performance, Design.

Palavras-chave

Organização de memória, hierarquia de memória, memória cache, instruções.

1. INTRODUÇÃO

A organização dos processadores de alto desempenho é dividida em um mecanismo de busca de instruções (*instruction fetch*) e um mecanismo de execução dessas instruções. Entre estes dois mecanismos podemos colocar *buffers* ou filas. O mecanismo de *instruction fetch* é responsável por buscar, decodificar e colocar as instruções no *buffer*. O mecanismo de execução é responsável por remover as instruções do *buffer* e executá-las, de acordo com a disponibilidade de dados e de recursos.

Observando as instruções que estão no *buffer* podemos identificar e explorar paralelismo em nível de instrução (ILP) e executar várias instruções no mesmo ciclo. Para tirarmos vantagem disso temos que ter o máximo possível de instruções neste *buffer*.

Não teremos problemas se colocarmos neste *buffer* todas as instruções do mesmo bloco básico, uma vez que todas as instruções desses blocos devem ser executadas. Por definição em um bloco básico só teremos o desvio no final do mesmo, e se não tivermos conflitos de dados elas podem ser executadas em paralelo, obtendo maior desempenho. A unidade de ILP, tomando cuidado com conflito de dados e recursos, identifica em um bloco básico quais instruções podem ser executadas em paralelo e as encaminham pelo caminho de dados (*datapath*) para usarem unidades funcionais paralelas do mecanismo de execução. Com isto podemos executar um número maior de instruções por ciclo.

Na tabela 1, adaptada de [1], vemos que o bloco básico, para programas de inteiros, tem em média 4 ou 5 instruções, sendo assim não conseguimos, com apenas um bloco básico no *buffer* atingir um grande ILP. Podemos melhorar ILP se utilizarmos predição de *branchs*, colocando, em um mesmo ciclo, além do bloco básico atual o possível próximo bloco básico no *buffer*. Se utilizarmos predição de múltiplos *branchs* por ciclo, colocamos múltiplos blocos básicos por ciclo no *buffer*. Conseqüentemente aumentamos o *throughput* de instruções, pois estamos colocando um número maior de instruções para serem analisadas e possivelmente executadas em paralelo a cada ciclo.

Tabela 1: Estatísticas de *branch* e blocos básicos

Benchmark	*Branch* tomados %	Tam. médio do bloco	Número de instruções entre *branch* tomados
Eqntott	86.2%	4.20	4.87
Espresso	63.8%	4.24	6.65
Xlisp	64.7%	4.34	6.70
Gcc	67.6%	4.65	6.88
Sc	70.2%	4.71	6.71
Compress	60.9%	5.39	8.85

Na seção seguinte explicaremos o que são e como funcionam *trace caches*. Na seção 3 mostraremos a implementação de *trace cache* no processador 'Pentium 4' da Intel. Na seção 4 temos a metodologia e os resultados de [1], mostrando a melhora no desempenho que temos com as *trace caches*. Na seção 5 temos a conclusão, seguida das referências na seção 6.

Este trabalho pode ser impresso, fotocopiado e distribuído para fins educacionais, sendo desnecessário autorização especial do autor por escrito.
Para outros fins favor entrar em contato diretamente com o autor.
A UNICAMP se isenta da responsabilidade do conteúdo deste trabalho, sendo esta única e exclusivamente do autor.

2. TRACE CACHES

A *trace cache* é uma *cache* suplementar a *cache* de instruções e tenta prover uma alta taxa de entrega de instruções ao decodificador de instruções do processador. Esse aguarda a próxima instrução da *trace* cache ou da *cache* de instruções convencional, o que ocorrer primeiro, e possivelmente executa um certo número de instruções em paralelo.

Na *cache* de instruções convencional temos as instruções na ordem de compilação, o que é favorável para programas que não tenham *branch*s tomados ou blocos básicos grandes. Porém como vimos na tabela 1 isto não é o que comumente ocorre em programas de inteiros.

A *trace cache* procura explorar a seqüência dinâmica das instruções. Esta seqüência, ou *trace*, tem no máximo *n* instruções e *m* blocos básicos. Cada linha da *trace cache* tem, com o auxílio dos preditores de *branch,* um conjunto grande de instruções, organizados pelo endereço da primeira instrução do *trace* (*Program Counter - PC*) e *m bits* (que é a capacidade do preditor de *branch)* indicando pra qual predição de *branch* (tomados/não tomados) aquele *trace* corresponde, ver figura 1, adaptada de [1].

A *trace cache* vai sendo preenchida conforme o andamento do programa, a cada *trace* diferente encontrado uma nova linha é utilizada. Note que para um mesmo PC podemos ter mais de uma linha se a saída do preditor de *branch* for diferente. Algumas implementações transformam isso em duas linhas na *trace cache* com mesmo PC, mas com *bits* diferentes representando a seqüência de *branchs* tomados/não tomados. Outras implementações permitem que apenas uma linha tenha um determinado PC e nesse caso substitui o *trace* antigo pelo novo.

Se o mesmo *trace* for encontrado novamente (mesmo endereço da primeira instrução e a mesma saída do preditor de *branch*) a *trace cache* entrega todas as instruções do *trace* para o decodificador, como ilustrado na figura 1. Caso contrário quem entrega ao decodificador é a *cache* de instruções convencional.

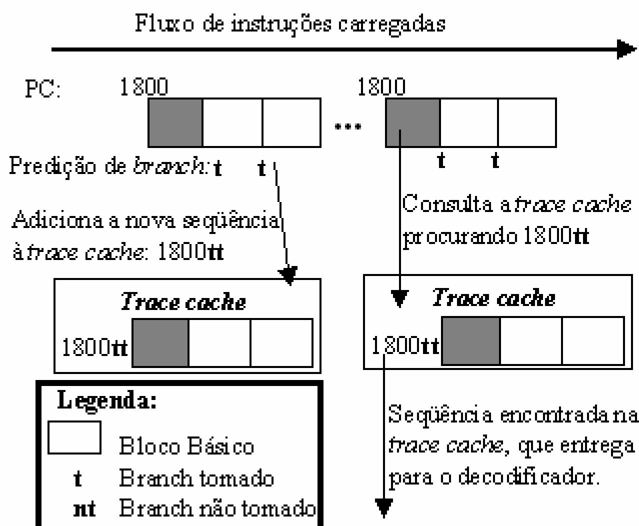

Figura 1: Visão geral do comportamento da *trace cache*

Do valor de *m* e do tamanho total da *trace cach*e derivamos o valor de *n*, tomando o cuidado para reservar alguns *bits* por linha da *trace cache* para o controle da mesma.

Um bom preditor de *branch* é essencial para que a *trace cache* seja eficiente pois a cada predição errada, assim como já ocorria na *cache* de instruções convencional, vários ciclos não são aproveitados, com execução do nosso programa, até que a próxima instrução seja carregada da memória para a *cache* e dessa para o decodificador do processador.

Assim como outras *caches,* a *trace cache* tem uma capacidade de memória limitada e é preciso estabelecer algum critério para substituir as linhas de *trace*. Para isso mais alguns bits devem ser incluídos a cada linha, por exemplo, *valid bit* assinalando quais traces são válidos e usar algum algoritmos de substituição comum em cachês, como *LRU* ou *Round Robin*.

3. TRACE CACHE NO PENTIUM 4

A Intel usa em seus processadores baseados na micro-arquitetura *NetBurst*, dentre eles o Pentium 4, a *trace cache* como *cache* de instruções. Reduzindo o tempo de busca das instruções, ao custo de um *hardware* um pouco mais complexo [2].

Esses processadores executam micro-operações que são derivadas do conjunto de instruções IA-32, e por sua característica *superscalar* conseguem executar até três instruções IA-32 em um ciclo de clock.

A tradução de uma instrução IA-32 para micro-operações anteriormente era feita por PLAs (*Programmable logic arrays*) que analisavam até três instruções e geravam até seis micro-operações por ciclo, porém com a alta freqüência de *clock* do Pentium 4 PLAs se tornaram inviáveis [3].

A solução encontrada foi utilizar uma *trace cache* que armazena seqüências de micro-operações, auxiliada por uma ROM para lidar com instruções mais complexas, que necessitam de um número maior de micro-operações para representar a instrução IA-32, de tal forma que essa ROM complementa a seqüência de micro-operações que representam a instrução IA-32 que deve ser decodificada.

Comparada com uma *cache* de instruções convencional, a grande vantagem da *trace cache* é que sua linha é preenchida em um ciclo com instruções de mais de um bloco básico, o que não acontece na *cache* convencional que armazena instruções de um único bloco básico até a instrução de *branch*, que pode ocorrer muito precocemente, por exemplo, um bloco básico pode ter apenas uma instrução, a de *branch*.

Uma vez que um *trace* já foi decodificado e está na *trace cache*, temos em laços, por exemplo, a repetição deste mesmo *trace* e com isto removemos a latência em decodificar as instruções do laço. A *trace cache* entrega as instruções já em micro-operações, prontas para entrar no *pipeline* de execução.

Uma *trace cache* de instruções nível 1 do Pentium 4 é capaz de armazenar até 12000 micro-operações decodificadas, com uma taxa de acerto (*hit rate*) similar a uma *cache* de instruções convencional com tamanho entre 8kb e 16kb. Se tivermos mais de um processador lógico é colocado um marcador para indicar qual *trace* é de qual processador [4].

4. MELHORA NO DESEMPENHO

Em processadores superescalares de alto desempenho, conseguimos executar várias instruções em paralelo, pois várias unidades funcionais são duplicadas. Sendo assim quanto mais unidades funcionais conseguirmos estar utilizando ao mesmo tempo em nossa computação, melhor será o desempenho.

Para conseguirmos ter sempre um conjunto de instruções que possivelmente ocupem estas unidades funcionais é necessário buscar e decodificar várias instruções por ciclo de *clock*. A *cache* de instruções convencional começou a se tornar um gargalo nesses processadores e as *trace caches* mostraram ser uma alternativa inteligente pois além de diminuir o tempo de espera na decodificação de instruções, é capaz de entregar vários blocos básicos, conseqüentemente várias instruções, por ciclo de *clock*.

Na proposta original de Rotenberg [1], este mostra uma melhora média no desempenho de 28% em *benchmarks* com inteiros. Através de um simulador ele faz a comparação alguns mecanismos de *fetch*: SEQ.1, SEQ.3, BAC, CB e TC.

SEQ.1 e SEQ.3 respectivamente buscam um ou três blocos básicos contínuos por ciclo. BAC quer dizer *Branch Address Cache* e está explicado em [6]. CB é *Collapsing Buffer* e está explicado em [7]. TC é *trace cache*.

A *trace cache* mostrou melhor desempenho nos *benchmarks* de inteiros (SPEC92 e IBS) e ainda observa que é possível melhorar ainda mais a *trace cache*, por exemplo aumentando seu tamanho e/ou sua associatividade.

Em [5] Sung também através de um simulador comprova a superioridade da *trace cache* e aplica os testes sobre o *benchmark* SPEC95 tanto para inteiros quanto para ponto-flutuante. Como ilustra a figura 2, também de [5], a comparação entre a *cache* de instruções convencional e a *trace cache* é medida através do número de *fetch* de instruções por ciclo de *clock* (IPC), quanto maior melhor. A *trace cache*, colunas cinza na figura 2, em todos os *benchmarks* se foi melhor, com um ganho médio de 17%, incluindo testes com inteiro e com ponto-flutuante.

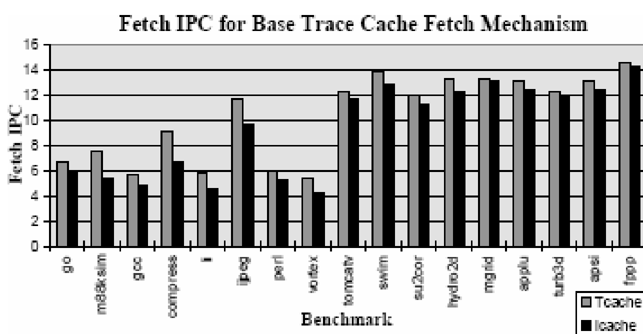

Figura 2: Ganho de desempenho da *trace cache* sobre a *cache* de instruções convencional.

Além disso, ele fez algumas melhorias na indexação e comparou o efeito do ajuste dos seguintes parâmetros: tamanho de cada linha entre 10 e 24 instruções, número de *branchs* previstos entre 1 e 5, mudanças na política de preenchimento da *trace cache*, acertos parciais dos *branchs* da seqüência, aumento da associatividade entre diretamente (1 via), 2 vias, 4 vias e 8 vias, tamanho total da *trace cache* ou número de linhas dessa entre 64 e 4096 e por último a política de substituição de linhas da cache (aleatória, LRU ou *round robin*).

Com esses parâmetros otimizados uma *trace cache* chega a ter um desempenho em *fetch* de instruções correspondente a 67.9%, para inteiros, e 16.3%, para ponto-flutuante, melhor do que uma *cache* de instruções tradicional.

Destes parâmetros chamamos atenção para acertos parciais dos *branchs* da seqüência, que sozinho representa uma melhora de 9.4% na média para a menor *trace cache*. Para podermos utilizar desse recurso temos que, em cada linha da *trace cache,* adicionar o endereço de cada bloco básico que tem naquela seqüência e um ponteiro para o final de cada bloco básico.

Agora quando formos procurar por um *trace* comparamos o endereço inicial e a saída do preditor de *branch* com os *bits* que representam a ordem dos *branchs* tomados/não tomados.

Como aceitamos acertos parciais se, por exemplo, a primeira predição está correta mas a segunda não, entregamos para o decodificador do processador os dois primeiros blocos básicos. A figura 3, adaptada de [5], ilustra este exemplo. Note que utilizamos o ponteiro do segundo bloco básico para sabermos até qual instrução devemos entregar.

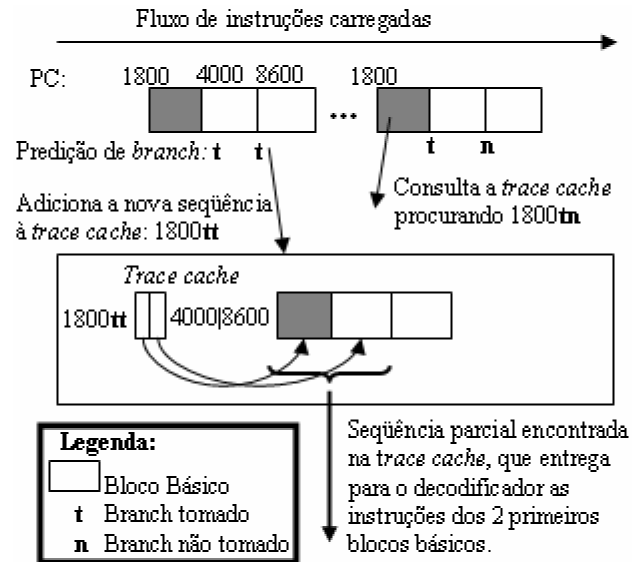

Figura 3: Estruturas adicionais e exemplo para acerto parcial da *trace cache* com o preditor de *branch*.

Na figura 4 temos destacado o ganho de desempenho usando acerto parcial. Assim como na figura 2, a comparação é feita em IPC e como base temos a *trace cache* em sua configuração básica, colunas pretas. As colunas mais claras são os desempenhos usando *trace cache* com acerto parcial.

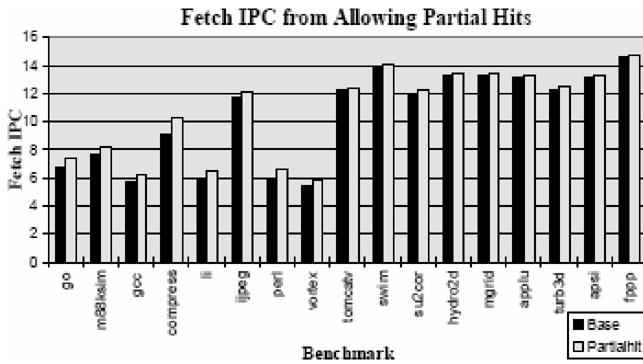

Figura 4: Ganho de desempenho usando acerto parcial

5. CONCLUSÃO

As *trace caches* mostraram ser uma alternativa eficaz à cache de instruções para os processadores superescalares, que necessitam de um grande número de instruções decodificadas por ciclo de *clock,* para apresentarem ganho de desempenho significativo na execução em paralelo de um conjunto de instruções.

Quanto melhores forem os preditores de *branch* maior será o ganho de desempenho que as *trace caches* proporcionam, uma vez que esta é fortemente dependente da saída do preditor para poder armazenar e futuramente entregar para o processador mais instruções de blocos básicos que serão executados em seqüência. Essa dependência não é preocupante, pelo contrário, é grande o número de trabalhos na área de predição de *branchs*, incluindo a predição de múltiplos *branchs* e os resultados são muito bons.

A implementação comercial das *trace caches* nos processadores da Intel, como o Pentium 4, substitui a *cache* de instruções por uma *trace cache* que armazena seqüências de micro-operações.

O desempenho superior da *trace cache* sobre uma *cache* de instruções convencional e sobre outros mecanismos de *fetch* de instruções dá motivação para que *trace caches* sejam estudadas. Ainda há melhorias que estão sendo feitas, por exemplo, fazê-las gastarem menos potência para serem viáveis em dispositivos móveis [8].

6. REFERÊNCIAS

[1] Rotenberg, E., Bennett, S., Smith, J. E., "Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching." *29th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-29), p. 24-3, 1996.*

[2] J. Hennessy and D. Patterson, "Computer Architecture: A Quantitative Approach". Second Edition. Morgan Kaufmann, 1996.

[3] J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software interface". Thind Edition. Morgan Kaufmann, 2005.

[4] Kerly, P., "Cache Blocking Technique on Hyper-Threading Technology Enabled Processors". Artigo na página da Intel: http://www.intel.com/cd/ids/developer/asmo-na/eng/20461.htm.

[5] Sung, M., "Design of Trace Caches for High Bandwidth Instruction Fetching", 1997.

[6] T.-Y. Yeh, D. T. Marr, and Y. N. Patt., "Increasing the instruction fetch rate via multiple branch prediction and a branch address cache". *7th ACM International Conference on Supercomputing, pp. 67-76, July 1993*.

[7] T. Conte, K. Menezes, P. Mills, and B. Patel. "Optimization of Instruction Fetch Mechanisms for High Issue Rates". *22nd International Symposium on Computer Architecture, pp. 333-344, June 1995*.

[8] Chaver, D., Rojas, M. A., Pinuel, L., Prieto, M., Tirado, F., Huang, M.C., "Energy-aware fetch mechanism: trace cache and BTB customization", *International Symposium on Low Power Eletronics and Design, pp. 42-47, 2005.*