Expansão de atividades do projeto3

Paulo H. Ribeiro[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicam)
Campinas – SP – Brazil

paulohsilvar@gmail.com

Abstract. *This documentation comes to describe the project4 of the discipline MO601 of the Computing Institute of Unicamp. The work consists of expanding the activities carried out in the previous project. This new work compares the results already obtained with new fetching mechanisms in the processor pipeline. As result a graph is presented that compares the IPC reached in each mechanism*

Resumo. *Esta documentação de reporte vem para descrever o projeto4 da disciplina MO601 do Instituto de Computação da Unicamp. O trabalho consiste de expandir as atividades realizadas no projeto anterior. Esse novo trabalho compara os resultados já obtidos com novos mecanismos de fetching no pipeline do processador. Como resultado é apresentado um gráfico que compara o IPC alcançado em cada mecanismo.*

1. Introdução

No projeto anterior, foi escolhido o artigo Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching, para reprodução do gráfico da Figura8. Ele introduz o conceito de trace cache visando aumentar o bandwidth de instruções na etapa de fetching do pipeline. O problema que o artigo busca solucionar é o fato de que as instruções são gravadas na cache de instrução convencional na ordem em que foram compiladas, fazendo com que instruções sejam carregadas na cache e não sejam usadas, por causa de branchs por exemplo. A solução apresentada é o que ele chama de Trace Cache, ela corresponde a uma cache para instruções de forma que elas sejam armazenadas em ordem dinâmica, visando seguir a sequencia que o programa toma durante sua execução.

Com essa nova cache estaremos evoluindo nosso mecanismo de fetching, pois o objetivo é que os componentes já existentes como a cache de instrução o hardware para os preditores não sejam replicados

Como objetivo deste novo projeto buscamos expandir os resultados apresentados no projeto anterior. Vamos adicionar novas colunas no gráfico de resultado, representado o uso de um brand predictor de 1 bit. Buscando melhorar os resultados alcançados anteriormente, algumas modificações de implementação foram necessárias.

2. Ferramentas utilizadas

A ferramenta utilizada para o desenvolvimento dessa atividade foi o simulador

de processador Sniper. O que motivou a escolha dele foi o fato de que ele é baseado no PIN Instrumentation Tool, ferramenta essa que foi utilizada nos projetos anteriores.

Esse simulador possui na versão 6.1 (versão mais atual) um package que permite simular realizar suas simulações usando alguns Benchmarks, como exemplo o Splash2.

3. Desenvolvimento

Até o final do desenvolvimento do projeto4 não houve nenhuma resposta dos desenvolvedores do artigo utilizado sobre suporte para realização da tarefa.

Tendo um ambiente inicial de Trace Cache já implementado no simulador, que foi usado no projeto3, alguns novos ajustes foram realizados. Os níveis de cache foram alterados, nessa nova etapa a Trace Cache possui a Cache de Instrução – L1 como sua sucessora na hierarquia, diferentemente da implementação anterior quando ela não tinha nenhuma sucessão. Foram ajustados assim a forma de realizar os acessos as caches, pois agora todas as caches possuem níveis seguintes.

Um novo ajuste realizado no simulador foi para apresentar nos arquivos de saída os resultados dos contadores da Trace Cache. Alguns objetos precisaram ser alterados, como o MemComponent, por exemplo. Essa classe contém um enum que caracteriza qual o tipo de componente há dentro do processador. Foi preciso criar um novo para a Trace Cache, de forma que mantivemos a Cache de Instrução – L1 como sendo o primeiro nível de cache para o processador, pois mesmo sendo ela sucessora da Trace Cache, ela não foi para o segundo nível, nela que são carregada as instruções lidas da memória. Foi necessário também ajustar os scripts em python que escrevem nos arquivos de saída para reconhecerem a Trace Cache e ler suas informações e escrevê-las.

Foi desenvolvido também um script em python capaz de ler os arquivos de resultado e gerar um gráfico com configuração similar a Figura8 do artigo original.

4. Resultado

A Figura1 a seguir, apresenta o resultado de IPC alcançado para o Benchmark FFT do Splash2. Os cenários de simulação foram:

1. Fetching já implementado no simulador;
2. Fetching com o brand predictor de 1 bit;
3. Fetching com Trace Cache e brand predictor original do simulador;
4. Fetching com Trace Cache e brand predictor de 1 bit.

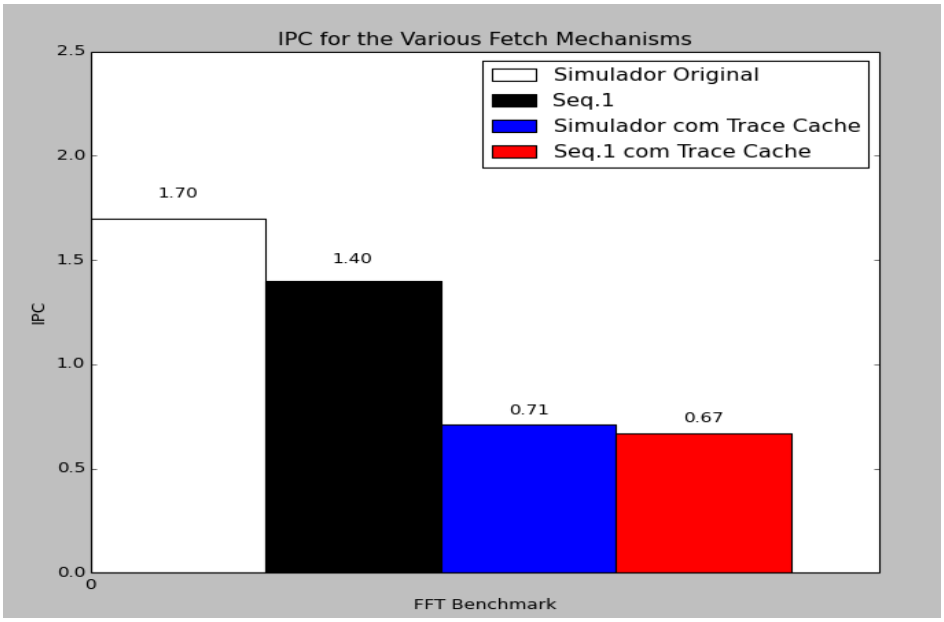


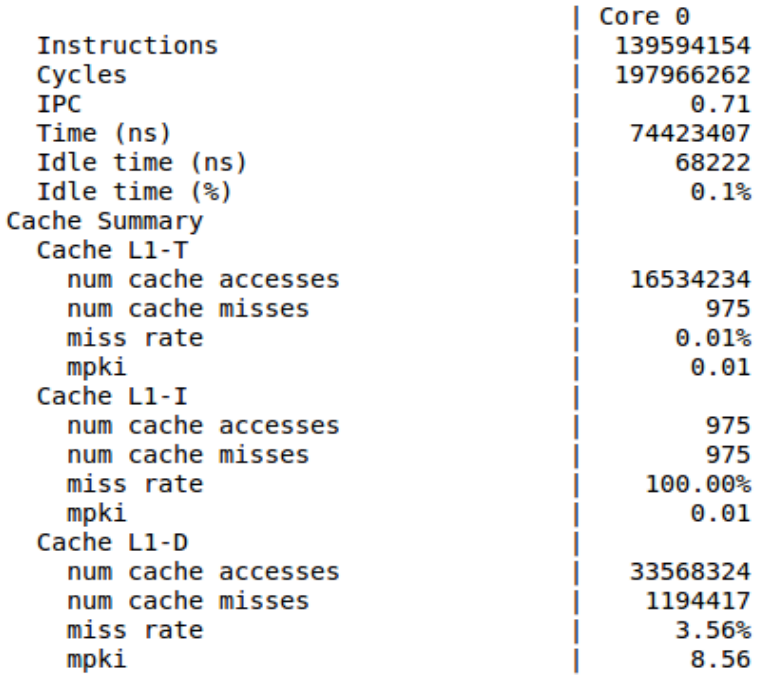
Figura 1. IPC

Podemos perceber que todos os resultados usando o brand predictor de 1 bit se saíram em pior caso comparando ao mesmo cenário com o brand predictor original do simulador. O principal resultado observado é o fato de que não conseguimos alcançar o ganho de IPC apresentado pelo artigo usando esse novo mecanismo de Fetching.

Buscando encontrar uma explicação para a diferença de IPC usando ou não o Trace Cache foi analisado os hits/misses nas caches, os resultados estão apresentados na Figura2 e Figura3 a seguir.

	Core 0
Instructions	139594155
Cycles	82113631
IPC	1.70
Time (ns)	30869786
Idle time (ns)	67418
Idle time (%)	0.2%
Cache Summary	
Cache L1-I	
num cache accesses	16534234
num cache misses	925
miss rate	0.01%
mpki	0.01
Cache L1-D	
num cache accesses	33568324
num cache misses	1194275
miss rate	3.56%
mpki	8.56

Figura 2. Resultado hit/miss sobre Caches L1 sem usar Trace Cache no Fetching.



	Core 0
Instructions	139594154
Cycles	197966262
IPC	0.71
Time (ns)	74423407
Idle time (ns)	68222
Idle time (%)	0.1%
Cache Summary	
Cache L1-T	
num cache accesses	16534234
num cache misses	975
miss rate	0.01%
mpki	0.01
Cache L1-I	
num cache accesses	975
num cache misses	975
miss rate	100.00%
mpki	0.01
Cache L1-D	
num cache accesses	33568324
num cache misses	1194417
miss rate	3.56%
mpki	8.56

Figura 3. Resultado hit/miss sobre Caches L1 usando Trace Cache no Fetching.

Com os resultados dos hits/misses e quantidade de acessos sobre as caches, podemos chegar a conclusões. Analisando apenas a Figura2 inicialmente, conseguimos capturar a quantidade de acessos a Cache de Instrução e a quantidade de Misses ocorridos. Percebemos que nesse cenário a Cache cumpriu seu papel de ajudar no desempenho, onde alcançamos um IPC de 1.7.

Quando inserimos o mecanismo de Trace Cache no estágio de Fetching do pipeline do processador, percebemos que os acessos que no cenário anterior ocorriam na Cache de Instrução – L1 estão ocorrendo na Trace Cache e que o número de Misses na Trace Cache é muito próximo da quantidade que a Cache de Instrução – L1 sofreu anteriormente e igual ao que ela sofre neste cenário.

Percebe-se que a Trace Cache está recebendo e despachando as suas instruções, tendo em vista que poucos Misses estão ocorrendo nela. Podemos concluir que um possível motivo para o valor de IPC apresentado é de que a Trace Cache e a Cache de Instrução – L1 estejam funcionando de forma sequencial e não paralela. Desta forma, todos os Misses que ocorrem gasta-se o dobro de tempo para acessar mais um nível sequencialmente, prejudicando assim o desempenho do processador.

5. Conclusões

Não temos dúvidas conforme apresentado pelo artigo original de que o mecanismo de Trace Cache no estágio de Fetching ajuda no ganho de desempenho do processador. Embora sem a ajuda dos autores do artigo para implementação do mecanismo, não conseguimos alcançar os mesmos resultados que eles. Contudo com a implementação desse mecanismo foi possível aprofundar os conhecimentos sobre processador adquiridos para uma forma mais prática, tendo que adicioná-lo dentro de um simulador para obter resultados de desempenho.

6. References

Sniper,

http://snipersim.org/documents/sniper-manual.pdf

Wikipedia,

https://en.wikipedia.org/wiki/CPU_cache