## Universidade Federal do Amazonas - UFAM

Instituto de Computação - IComp

**Professor: Juan Colonna** 

**Disciplina:** ICC062 - Arquitetura de Computadores

**Aluno:** Victor Hugo de Oliveira Gomes - 21852452

## Relatório - EduMIPS64

 Qual foi o maior gargalo no desempenho? Onde o código sofreu a maior perda de desempenho?

O maior gargalo de desempenho do código foi nos branches e nos RAWs na hora de usar os valores que estavam sendo lidos das matrizes. Como o EduMIPS não implementa (ou parece não implementar) branch prediction, todas as vezes que os laços eram avaliados, acontecia uma parada de pipeline, fazendo o programa perder 5 ciclos inteiros. Ao todo, o programa tem 936 RAWs, sem forwarding isso faz com o que o programa perca 3 ciclos por RAW, aumentando significativamente o CPI.

2. Por que o código teve o CPI que teve? Explicar.

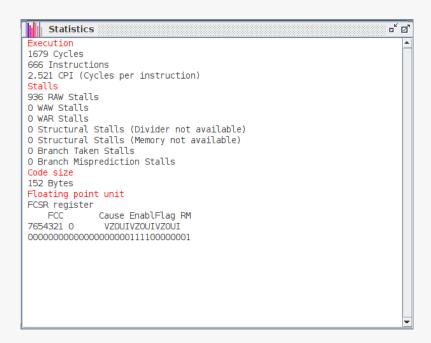


imagem 1. Estatísticas do código matrix.s sem forwarding

A imagem 1 mostra as estatísticas para a execução do código matrix.s. Apesar de ter poucas instruções, muitas instruções precisam utilizar os mesmos

registradores que as instruções acima estão utilizando, criando uma dependência de registradores entre instruções subsequentes causando aumento da quantidade de ciclos para um número fixo de instruções e, consequentemente, aumentando o CPI. Podemos ver na imagem abaixo, as estatísticas para o mesmo código, porém com forwarding ativado.

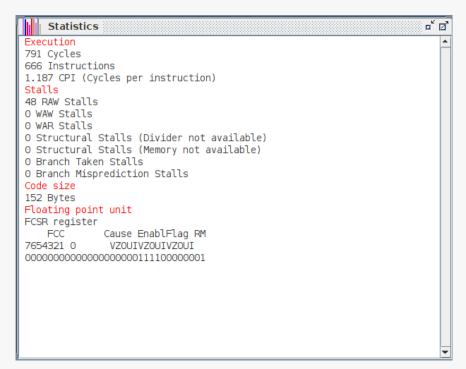


imagem 2. Estatísticas do código matrix.s com forwarding

Já podemos ver a drástica mudança. Mesmo com a restrição de registradores, a quantidade de RAWs diminuiu cerca de 95%, porque agora os valores podem ser acessados antes da sua escrita na memória.

3. Como poderia-se melhorar o programa? As restrições exigidas interferiram no desempenho do código? Explique o porquê.

A primeira coisa a se fazer, como discutido na questão anterior, é ativar forwarding. Isso vai acabar com a maioria das dependências de registradores que o código tem, diminuindo significativamente o CPI. A Segunda coisa é analisar o que está causando os RAWs restantes. 48 RAW Stalls, 2 por iteração no laço mais interno. O primeiro acontece quando é preciso multiplicar os registradores t0 e s7, que contém, respectivamente, os valores nos índices das matrizes b e a.

Como logo antes da multiplicação ainda estamos gravando no registrador t0 o valor da matriz b, ainda não temos o seu valor para realizar a multiplicação, então a instrução de MULT fica parada 1 ciclo.

O segundo RAW Stall acontece, de forma similar, porque ainda estamos buscando o valor guardado na matriz, nesse caso, a matriz c e logo em seguida, somamos o valor do registrador s6, com t0 (que contém o valor da multiplicação de a com b) para em seguida guardar o valor resultante de volta na matriz c.

O stall descrito por último pode ser resolvido apenas mudando a instrução que atualiza o índice do loop mais interno de posição. Podemos mudar para a instrução que atualiza s2 para antes da soma entre t0 e s6, isso vai fazer com que o processador não gaste mais um ciclo parado, pois vai estar fazendo essa soma e de quebra, teremos o valor para guardar s6 de volta em c sem causar um RAW stall.

O primeiro stall não é tão simples de resolver e só consegui resolver com a adição de mais um registrador, para guardar o valor do índice de c, sem precisar usar o mesmo registrador que está sendo usado por outra operação.

Existem ainda outras melhorias que fazem com que o número de instruções diminua, por exemplo, utilizar SLL ao invés de MULT onde é possível. Essa instrução executa um bit shift à esquerda de uma quantidade fixa representada por um imediato. Toda instrução de MULT, dentro do EduMIPS, tem que ser seguida por uma instrução de mflo, para mover o valor da multiplicação que está armazenada nesse registrador, para algum outro registrador. Ao usarmos SLL, nos livramos da necessidade desta segunda instrução, diminuindo, assim, a quantidade total de instruções do código.

Talvez seja possível também, ao invés desses cálculos com índices feitos nesse código, utilizar um endereço diretamente como os índices das matrizes, nos livrando da necessidade das multiplicações.

As melhorias discutidas, com exceção da última, podem ser encontradas no código matrix2.s

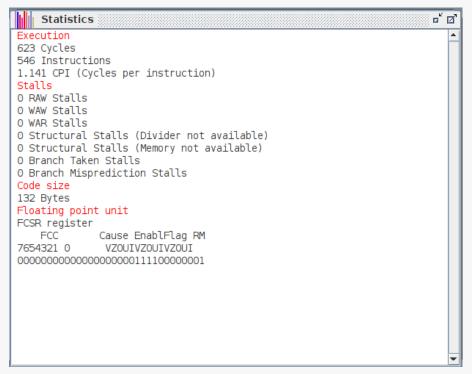


imagem 3. Estatísticas do código matrix2.s com forwarding

Como pode ser observado, apenas a adição de mais um registrador, acabou com todos os RAW stalls. A adição desse registrador tornou irrelevante a mudança de posição da instrução que atualiza o índice do loop mais interno.