UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO LUIZ GRAVE GROSS - 180171

TRABALHO PRÁTICO INDIVIDUAL 01

Trabalho da Disciplina Organização de Computadores B

Prof. Philippe O. A. Navaux

Porto Alegre, 29 de setembro de 2011.

1 INTRODUCAO

O presente trabalho consiste na execução de três benchmarks (MM, CRC e GO) em um simulador da arquitetura MIPS (simplescalar). Através das simulações, foram obtidos o IPC e o número de ciclos em cada execução.

Tais execuções tiveram como objetivo a avaliação do impacto da variação de predição de desvios e de grau de superescalaridade.

Um pouco de fundamentação a respeito de como as execuções foram feitas e do ambiente de trabalho são apresentadas no capítulo 2. O capítulo 3 contém os resultados obtidos e sua análise. Por fim, no capítulo 4, a conclusão.

2 AMBIENTE DE TRABALHO, BENCHMARKS E EXECUÇÕES DOS TESTES

Este capítulo tem por objetivo detalhar algumas informações acerca do ambiente de trabalho utilizados para a execução dos testes em cada benchmark, bem como sobre os benchmarks e como foi organizada a execução dos testes.

2.1 Ambiente de Trabalho

O detalhes acerca do ambiente utilizado para realizar as execuções pode ser visto na imagem que segue:

Figura 1: ambiente de trabalho

2.2 Benchmarks e Configurações

No total foram utilizados 3 benchmarks, MM (multiplicação de matrizes), CRC (algoritmos de detecção de erros) e GO (simulação de um confronto entre dois jogadores no jogo GO). A seguir está a distribuição de intruções que cada benchmark executa:

Tabela 1: Distribuição de instruções em cada benchmark

	MM	**CRC**	**GO**
Load	898317 **(11.66%)**	12988 **(21.49%)**	3090393 **(18.87 %)**
Store	146089 **(1.90%)**	10815 **(17.89%)**	892851 **(5.45 %)**
Desvio Incondicional	150519 **(1.95%)**	4567 **(7.56%)**	475522 **(2.90%)**
Desvio Condicional	136344 **(1.77%)**	5412 **(8.95%)**	2167712 **(13.23%)**
Aritmética de Inteiros	6371210 **(82.72%)**	26661 **(44.10%)**	9753738 **(59.55%)**

Na execução de cada benchmark, cada quesito se cercou de configurações diferentes, para testar a resposta da arquitetura frente diferentes situações. A comprrensão dessas configurações é pertinente para que posteriormente possamos compreender os resultados obtidos, bem como sua análise.

- Quesito Preditor de Desvios
  - Configuração 1: Nunca toma o desvio (not taken);
  - Configuração 2: Máquina de estados de dois bits (bimodal);
  - Configuração 3: Nunca erra uma predição de desvio (perfect).
- Quesito Grau de Superescalaridade
  - Configuração 1: Uma unidade de inteiros;
  - Configuração 2: Duas unidades de inteiros;
  - Configuração 3: Quatro unidades de inteiros.

2.3 Execução dos Testes

Quanto às execuções, realizou-se 6 execuções para cada benchmark, sendo 3 para preditores de desvio nas configurações nottaken, bimod e perfect e outras 3 para superescalaridade, utilizando 1, 2 e 4 ULAs, No total foram feitas 18 execuções, pois foram testados 3 benchmarks.

Para a execução dos testes utilizou-se um comando no terminal:

- Preditores de Desvio:
  $ ./sim-outorder -bpred <nome_do_preditor> -redir:sim <saida_simulador.txt> ./benchmarks/<comando_final>

  Onde:
  - <nome_do_preditor>: nottaken, bimod ou perfect;
  - <saida_simulador.txt>: nome do arquivo para salvar o resultado do teste;
  - <comando_final>: mm.ss, crc.ss ou go.ss 1 8, cada um caracterizando um benchmark.

- Grau de Superescalaridade:
  $ ./sim-outorder -res:ialu <num_alu> -redir:sim <saida_simulador.txt> ./benchmarks/<comando_final>

  Onde:
  - <num_alu>: corresponde ao número de unidades lógicas e aritméticas da arquitetura superescalar. O valor pode ser setado para 1, 2 ou 4.

Visando otimizar as execuções criou-se um script, que executa os 18 scripts de uma só vez e cria os arquivos de cada execução.

```
#!/bin/sh

#Predição de desvio
#MM benchmark
simplesim-3.0/./sim-outorder -bpred nottaken    -redir:sim logs/mm_nottaken.txt     simplesim-3.0/benchmarks/mm.ss
simplesim-3.0/./sim-outorder -bpred bimod       -redir:sim logs/mm_bimod.txt        simplesim-3.0/benchmarks/mm.ss
simplesim-3.0/./sim-outorder -bpred perfect     -redir:sim logs/mm_perfect.txt      simplesim-3.0/benchmarks/mm.ss

#CRC benchmark
simplesim-3.0/./sim-outorder -bpred nottaken    -redir:sim logs/crc_nottaken.txt    simplesim-3.0/benchmarks/crc.ss
simplesim-3.0/./sim-outorder -bpred bimod       -redir:sim logs/crc_bimod.txt       simplesim-3.0/benchmarks/crc.ss
simplesim-3.0/./sim-outorder -bpred perfect     -redir:sim logs/crc_perfect.txt     simplesim-3.0/benchmarks/crc.ss

#GO benchmark
simplesim-3.0/./sim-outorder -bpred nottaken    -redir:sim logs/go_nottaken.txt     simplesim-3.0/benchmarks/go.ss 1 8
simplesim-3.0/./sim-outorder -bpred bimod       -redir:sim logs/go_bimod.txt        simplesim-3.0/benchmarks/go.ss 1 8
simplesim-3.0/./sim-outorder -bpred perfect     -redir:sim logs/go_perfect.txt      simplesim-3.0/benchmarks/go.ss 1 8

#Superescalaridade
#MM benchmark
simplesim-3.0/./sim-outorder -res:ialu 1  -redir:sim logs/mm_alu1.txt   simplesim-3.0/benchmarks/mm.ss
simplesim-3.0/./sim-outorder -res:ialu 2  -redir:sim logs/mm_alu2.txt   simplesim-3.0/benchmarks/mm.ss
simplesim-3.0/./sim-outorder -res:ialu 4  -redir:sim logs/mm_alu4.txt   simplesim-3.0/benchmarks/mm.ss

#CRC benchmark
simplesim-3.0/./sim-outorder -res:ialu 1  -redir:sim logs/crc_alu1.txt  simplesim-3.0/benchmarks/crc.ss
simplesim-3.0/./sim-outorder -res:ialu 2  -redir:sim logs/crc_alu2.txt  simplesim-3.0/benchmarks/crc.ss
simplesim-3.0/./sim-outorder -res:ialu 4  -redir:sim logs/crc_alu4.txt  simplesim-3.0/benchmarks/crc.ss

#GO benchmark
simplesim-3.0/./sim-outorder -res:ialu 1  -redir:sim logs/go_alu1.txt   simplesim-3.0/benchmarks/go.ss 1 8
simplesim-3.0/./sim-outorder -res:ialu 2  -redir:sim logs/go_alu2.txt   simplesim-3.0/benchmarks/go.ss 1 8
simplesim-3.0/./sim-outorder -res:ialu 4  -redir:sim logs/go_alu4.txt   simplesim-3.0/benchmarks/go.ss 1 8
```

Figura 2: Script para executar os 18 testes de performance

A execução do script se deu da seguinte maneira:

> cd <diretório onde encotra-se o script e a pasta simplesim-3.0 do simulador mips>

> sh run_benchmarks.sh

Todos os arquivos (18 no total) das execuções ficaram salvos na pastas /logs. Cada arquivo foi aberto em um editor de texto e deles foram extraídas as informações necessárias, ou seja, número de ciclos de execução e número de instruções por ciclo (IPC).

O capítulo 3 mostra os resultados obtidos, sua análise e comparação.

3 RESULTADOS

Dos resultados obtidos, abaixo podemos observar 6 tabelas, cada uma referente à execução de um benchmark em um quesito, preditor de desvios e grau de superescalaridade.

Tabela 2: Quesito de predição de desvios, benckmark MM

MM			
Preditor	**IPC (sim_IPC)**	**Número de Ciclos (sim_cycle)**	**Aumento de Desempenho**
not taken	1,6418	4691529	-
bimodal	2,0146	3823396	22,71%
perfect	2,0171	3818685	22,86%

Tabela 3: Quesito de grau de superescalaridade, benckmark MM

MM			
ULAs	**IPC (sim_IPC)**	**Número de Ciclos (sim_cycle)**	**Aumento de Desempenho**
1	1,0326	7459261	-
2	1,8283	4213052	77,06%
4	2,0146	3823396	95,10%

Tabela 4: Quesito de predição de desvios, benckmark CRC

CRC			
Preditor	**IPC (sim_IPC)**	**Número de Ciclos (sim_cycle)**	**Aumento de Desempenho**
not taken	0,7459	81269	-
bimodal	1,392	43546	86,62%
perfect	1,8137	33420	143,16%

Tabela 5: Quesito de grau de superescalaridade, benckmark CRC

CRC			
ULAs	**IPC (sim_IPC)**	**Número de Ciclos (sim_cycle)**	**Aumento de Desempenho**
1	0,8551	70888	-
2	1,2904	46973	50,91%
4	1,392	43546	62,79%

Tabela 6: Quesito de predição de desvios, benckmark GO

GO			
Preditor	**IPC (sim_IPC)**	**Número de Ciclos (sim_cycle)**	**Aumento de Desempenho**
not taken	0,6506	25176621	-
bimodal	1,015	16138920	56,01%
perfect	1,2028	13618743	84,88%

Tabela 7: Quesito de grau de superescalaridade, benckmark GO

GO			
ULAs	**IPC (sim_IPC)**	**Número de Ciclos (sim_cycle)**	**Aumento de Desempenho**
1	0,7261	22560143	-
2	0,964	16991879	32,76%
4	1,015	16138920	39,79%

Nas tabelas foi acrescentada mais uma coluna, referente ao aumento de desempempenho em relação ao IPC da primeira linha de cada tabela. Por exemplo, na tabela 2, a primeira linha consiste em um teste de predição de desvios e configuração nottaken. O aumento de desempenho associado às linhas bimodal e perfect, ambos, tomam como base o IPC da linha nottaken e comparam com o seu IPC, que tem como resultado a porcentagem de o quão mais eficiente foi o teste.

Já para os testes que utilizaram o quesito de grau de superescalaridade, assim como para o quesito de preditor de desvios, a primeira linha da tabela possui a quantidade de ULAs igual a 1 e um IPC associado. Nas linhas posteriores, com quantidade de ULAs igual a 2 e 4 respectivamente, o aumento de desempenho é calculado tomando como base o IPC da primeira linha (n° ULAs = 1) e o seu próprio IPC, resultando a porcentagem de aumento de desempenho descriminada nas tabelas.

O cáculo é feito da seguinte maneira:

Aumento de desempenho = $(IPC_i - IPC_1) / IPC_1$, onde:

- IPC_1 é o IPC da primeira linha da tabela, e
- IPC_i é o IPC da linha i, sendo i igual a 2 ou 3

3.1 Análise dos resultados

Nesta seção são analisados os resultados obtidos nos quesitos de preditor de desvios e de grau de superescalaridade, considerando os 3 benchmarks em cada quesito.

Visando facilitar a comparação dos resultados, alguns gráficos foram construídos.

3.1.1 Quesito Preditor de Desvios

Resultados obtidos utilizando-se o quesito preditor de desvios:

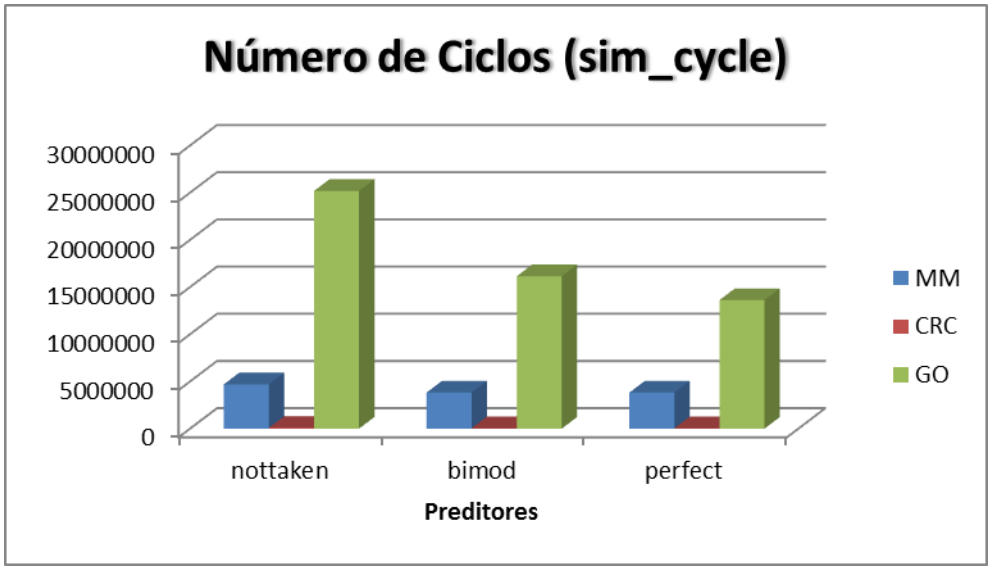

Figura 3: Número de ciclos para o quesito preditor de desvios

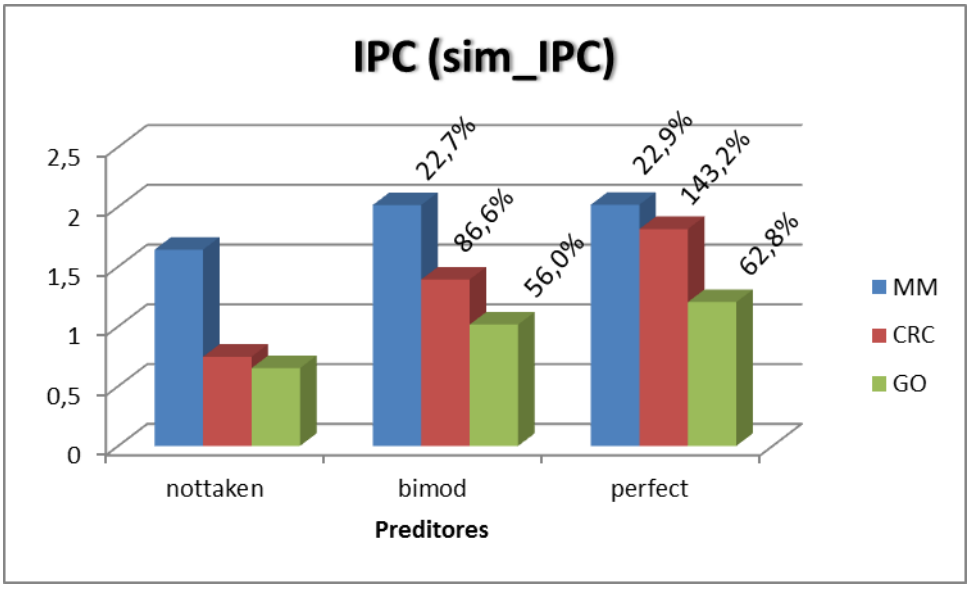

Figura 4: IPC para o quesito preditor de desvios

A primeira coisa que podemos observar, para os 3 benchmarks é que com a mudança de configuração na execução dos testes, de not taken, para bimod e porteriormente para perfect o desempenho do programa na arquitetura aumenta.

Isso ocorre, pois inicialmente, na configuração not token, toda a predição de desvio não é acertada, ou seja, a arquitetura processa algumas intruções que seriam executadas após a confirmação do desvio mesmo sem a confirmação do desvio. Só que como nessa configuração a predição nunca acerta, sempre são processadas instruções inutilmente, o que reflete em um número de ciclos maior e IPC baixo. A medida que a configuração muda, chegando a perfect, o desempenho aumenta, pois a predição sempre acerta, ou seja, as instruções que estavam sendo processadas na esperança de que o desvio iria ser verdadeiro, não precisam ser descartadas.

Já com relação ao aumento de desempenho de cada benchmark nos diferentes quesitos, percebemos que o ganho para o benchmark MM mudando as configurações foi pequeno, cerca de 20%. Isso se deve ao fato de que no MM são realizados menos desvios do que nos demais benchmarks, visto que a execução mais massiva de instruções é com operações aritméticas sobre inteiros.

Os benchmarks CRC e GO aumentaram seu IPC numa proporcão bem maior, pois diferente do MM ambos possuem mais instruções de desvio condicional.

3.1.2 Quesito Grau de Superescalaridade

Resultados obtidos utilizando-se o quesito grau de superescalaridade:

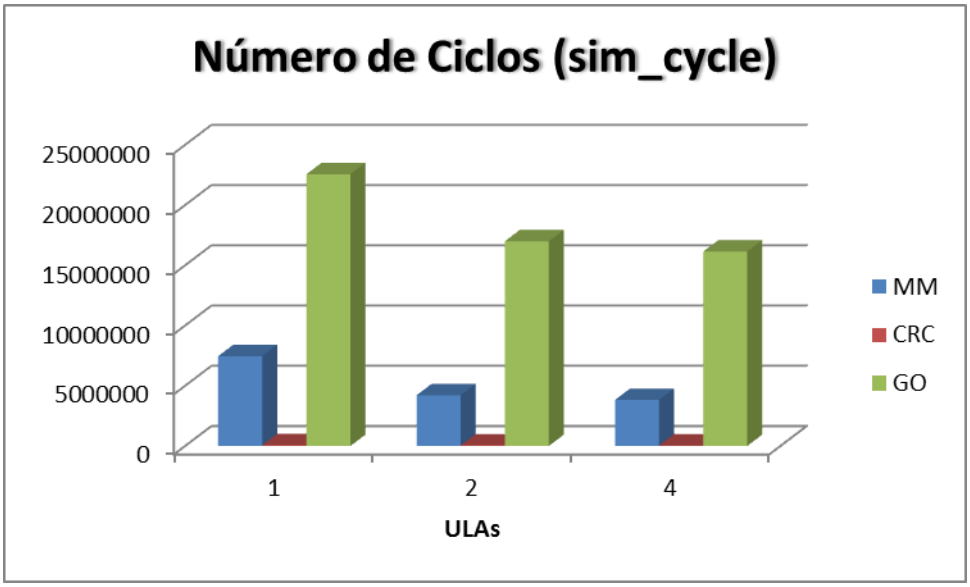

Figura 5: Número de ciclos para o quesito grau de superescalaridade

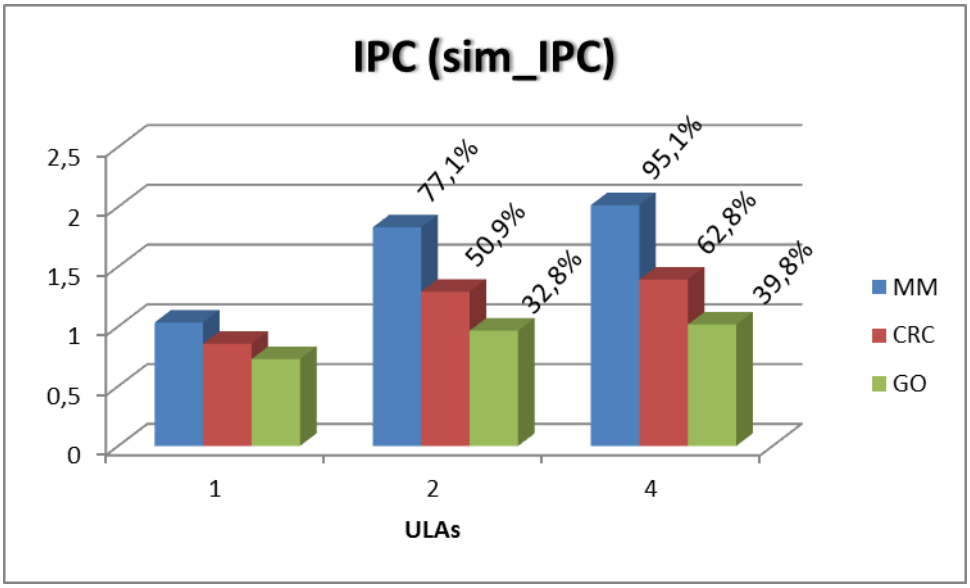

Figura 6: IPC para o quesito grau de superescalaridade

Observamos naturalmente, que assim como nas execuções utilizando o quesito preditor de desvios, o IPC tende a melhorar a medida que a configuração muda, mas aqui isso ocorre, pois o paralelismo da arquitetura aumenta e mais instruções são executadas simultaneamente.

O MM em particular obteve um desempenho melhor do que os demais benchmarks, pois ele possui mais operações aritméticas com inteiros, instruções essas de forte possibilidade de paralelização.

Entre o CRC e o GO, o CRC teve melhor desempenho pois possui uma distruição mais homogênia da quantidade de operações aritiméticas com inteiros, loads

e stores tirando mais proveito dos pipelines otimizados para executar cada uma dessas intruções.

4 CONCLUSÕES

Considerando os resultados obtidos pude perceber que um programa com rodando em uma arquitetua com predição de desvios eficiente sempre terá um bom desempenho. Já em arquiteturas com multiplas ULAs, quanto mais instruções puderem ser executadas simultaneamente melhor será o desempenho, logo o uso de mais ULAs trará deixará o programa mais rápido, porém é importante cuidar para que a arquitetura não fique muito cara, devido ao acréscimo de mais blocos de hardware.