Pontifícia Universidade Católica do Paraná
Escola Politécnica
Bacharelado em Sistemas de Informação

Disciplina: Performance em Sistemas Ciberfisicos
Professor: Guilherme Schnirmann

Nome Estudante: Ary Felipe Farah e Silva

Atividade Prática / Relatório

Desempenho – Exercícios

1) Considere duas máquinas com conjuntos de instruções diferentes. Os computadores tem uma frequência de 300 MHz. As medições a seguir foram registradas executando programas de um benchmark:

Tipo de instrução	Número de instruções (milhões)	Ciclos por tipo de instrução
Máquina A		
Aritmética e lógica	8	1
Load e store	4	3
Desvios	2	4
Outros	4	3
Máquina B		
Aritmética e lógica	10	1
Load e store	8	2
Desvios	2	4
Outros	4	3

a) Calcule o CPI médio para cada máquina
A: 8x1 + 4x3 + 2x4 + 4x3 = 40 milhões de ciclos / 18 milhões de instruções ~ 2.23
B: 10x1 + 8x2 + 2x4 + 4x3 = 46 milhões de ciclos / 24 milhões de instruções ~ 1.92

b) Calcule os tempos de execução
A - 40x~~10^6 ciclos~~
300x~~10^6 ciclos por segundo~~
40/300 = 0.14

B - 46x~~10^6 ciclos~~
300x~~10^6 ciclos por segundo~~
46/300 = 0.15

c) Calcule a taxa MIPS para cada máquina (pesquise o que é essa taxa).

MIPS - Milhões de Instruções por Segundo
Avalia a capacidade de processamento de uma máquina
MIPS = nº instruções executadas / tempo de execução em segundos
A – 18x~~10^6~~ / 0.14x~~10^6~~ = 18/0.14 ~ 128 MIPS
B – 24x~~10^6~~ / 0.15x~~10^6~~ = 24/0.15 ~ 160 MIPS

d) Sabendo que o desempenho de cada máquina é calculado por 1/(tempo de execução), calcule o desempenho de cada máquina e compare.
A = 1/0.14 = 7.14 execuções por segundo
B = 1/0.15 = 6.67 execuções por segundo

e) A razão entre os desempenhos é conhecida como ***speed-up.*** Calcule e interprete.
Indica quantas vezes uma máquina é mais rápida que a outra
Speed-up = Desempenho A / Desempenho B
s = 7.14 / 6.67 = 1.07 = a máquina A é aproximadamente 7% mais rápida que a máquina B, completando tarefas 1.7 vezes mais rapidamente.

2) Explique os três tipos de hazards em pipelines. Utilize exemplos para fundamentar suas respostas.
Hazard Estrutural – quando o hardware não suporta a execução paralela (ex: memória com uma porta só).
Solução: Atrasar o ciclo com conflito

Hazard de Controle – desvios / alterações de fluxo no programa, podendo fazer o pipeline carregar instruções desnecessárias ou fora de ordem.
Solução: Branch prediction / delayed branch

Hazard de Dados – Instrução depende do resultado de outra instrução que ainda está no pipeline, resultando em dependência de dados ainda não produzidos ou em trânsito
Solução: forwarding/bypassing

3) Pesquise, explique e exemplifique o que são as "bolhas" ou *stalls* no contexto de pipelining.
Bolhas (stalls) são pausas temporárias (ciclos de clock) no pipeline para resolver conflitos, como os hazards citados acima. Durante essa pausa, não são processadas novas instruções, mas sim um vazio, como uma bolha de ar.

4) Explique de forma objetiva o que são os computadores superescalares.
São sistemas de procesamento que possuem múltiplas unidades de execução, podendo executar várias instruções em paralelo desde que não exista dependência entre elas. Então, os computadores superescalares exploram o paralelismo a nível de instrução para aumentar o desempenho.

5) INTRODUÇÃO A THREADS EM JAVA

Em JAVA todas as propriedades e métodos estão encapsulados no corpo JDK. Assim, para criarmos uma thread instanciamos um objeto do tipo Thread. Passamos um objeto da classe que implementa a interface Runnable. O que inserirmos no método run será executado (desde que a thread seja agendada pelo S.O).

*Main.java

```java
public class Main {

    public static void main(String[] args) {
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                // código que irá rodar em uma nova thread
            }
        });

        thread.start(); //JVM cria a thread e passa para o S.O
    }
}
```

Replique o código (caso faça em python, pesquise como retornar o nome da thread). Atribua um nome a thread utilizando o método ***setName*().** Ainda, atribua uma prioridade utilizando ***setPriority().*** *Faça um teste utilizando a constante* ***Thread.MAXPRIORITY.*** Apresente o print dos seus testes com o nome dado e a prioridade.

```java
public class Main {
    public static void main(String[] args) {
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Thread rodando: " + Thread.currentThread().getName());
                System.out.println("Prioridade da thread: " + Thread.currentThread().getPriority());
            }
        });

        thread.setName("MinhaThread");
        thread.setPriority(Thread.MAX_PRIORITY);

        thread.start();
    }
}
```

```
Thread rodando: MinhaThread
Prioridade da thread: 10
```

Main.java

```java
public class Main {

    public static void main(String[] args) throws InterruptedException {
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Estamos na thread " + Thread.currentThread().getName());
            }
        });
        System.out.println("Estamos na thread " + Thread.currentThread().getName() + " antes de iniciar uma nova thread");
        thread.start(); //JVM cria a thread e passa para o S.O
        Thread.sleep(1000);
        System.out.println("Estamos na thread " + Thread.currentThread().getName() + " depois de iniciar uma nova thread");


    }
}
```

Uma outra maneira de implementarmos thread é criarmos uma classe e extendermos a classe Thread (que implementa Runnable). Agora conseguimos utilizar “this”.

```java

public class Main {

    public static void main(String[] args) {
        Thread thread = new MinhaThread();
        thread.start();
    }

    //Uma outra maneira de implementarmos thread é criarmos uma classe e extendermos a classe Thread (que implementa Runnable)
    private static class MinhaThread extends Thread {
        @Override
        public void run() {
            System.out.println("Olá da thread " + Thread.currentThread().getName());
        }
    }
}
```

Crie um programa em que duas threads são criadas (t1 e t2). No caso do JAVA, instancie dois objetos da sua classe Thread (MinhaThread) em um classe chamada Processo, por exemplo. Inicialize as duas threads e coloque um breakpoint no método run(). Observe e reporte quantos Threads existem.

```
Connected to the target VM, address: '127.0.0.1:56144', transport: 'socket'
Thread T2 está em execução.
Thread T1 está em execução.
Thread T2 - 0
Thread T1 - 0
Thread T2 - 1
Thread T1 - 1
Thread T2 - 2
Thread T1 - 2
Thread T2 - 3
Thread T1 - 3
Thread T2 - 4
Thread T1 - 4
Disconnected from the target VM, address: '127.0.0.1:56144', transport: 'socket'
```

Exercício:

Faça um programa que dado um intervalo (numérico), imprima todos os números primos existentes.

Exemplo:

Intervalo [2;10]

Primos: 2,3,5,7

a) Faça apenas o processo (sem threads)

```
Primos no intervalo [2-100]:
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
```

b) Criar um thread que imprime os primos no intervalo

```
Primos no intervalo [2-100]:
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 | 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
```

c) Criar 2 threads: Uma para primeira metade (começo até metade) e outra para (metade +1 até final).

```
Primos na primeira metade [2-51]:
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 |

Primos na segunda metade [52-100]:
| 53 | 59 | 61 | 67 | 71 | 73 | 79 | 83 | 89 | 97 |
```

d) Criar vários threads, um para cada sub-intervalo.
Exemplo: 5-50 ; 50-100; 100-150; 150-200

```
Primos no intervalo [2-25]:
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 |

Primos no intervalo [26-49]:
| 29 | 31 | 37 | 41 | 43 | 47 |
```

```
Primos no intervalo [50-73]:
| 53 | 59 | 61 | 67 | 71 | 73 |

Primos no intervalo [74-100]:
| 79 | 83 | 89 | 97 |
```

e) Faça testes com intervalos grandes (na casa de milhões). Qual dos exemplos anteriores é o caso mais rápido?
Início = 2
Fim = 1000000
Como vemos abaixo, o programa com duas threads foi o mais rápido.

f) Utilize o powershell para medir o tempo: *measure-command { java meu programa}. Para isso crie um programa para cada versão e meça o tempo para intervalos iguais.*

Primos:

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 581
Ticks             : 5814210
TotalDays         : 6,72940972222222E-06
TotalHours        : 0,000161505833333333
TotalMinutes      : 0,00969035
TotalSeconds      : 0,581421
TotalMilliseconds : 581,421
```

PrimosThread:

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 658
Ticks             : 6587957
TotalDays         : 7,62495023148148E-06
TotalHours        : 0,000182998805555556
TotalMinutes      : 0,0109799283333333
TotalSeconds      : 0,6587957
TotalMilliseconds : 658,7957
```

PrimosDuasThreads

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 526
Ticks             : 5264529
TotalDays         : 6,09320486111111E-06
TotalHours        : 0,000146236916666667
TotalMinutes      : 0,008774215
TotalSeconds      : 0,5264529
TotalMilliseconds : 526,4529
```

PrimosNThreads (4)

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 596
Ticks             : 5963245
TotalDays         : 6,90190393518518E-06
TotalHours        : 0,000165645694444444
TotalMinutes      : 0,00993874166666667
TotalSeconds      : 0,5963245
TotalMilliseconds : 596,3245
```