UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Curso de Ciência da Computação

**Processamento Paralelo e Distribuído**

Primeiro Trabalho

**Alunos:** Higor Borjaille Soneguetti

Bernardo Lucas Sunderhus

**Professor:** João Paulo Andrade Almeida

Vitória - ES

27 de Junho de 2016

**1 - INTRODUÇÃO**

Este trabalho foi desenvolvido para colocar em prática os conceitos de programação paralela e distribuída e ser realizado o checksum (sessão 1.3) de um vetor composto por bytes. Na implementação do trabalho foi explorada a arquitetura mestre-escravo que está detalhada na sessão 1.2 deste relatório.

Neste relatório serão apresentados testes distribuídos em duas ou mais máquinas, utilizados para comparar o tempo de execução, speedup e overhead para diferentes tamanhos de vetores com seus respectivos testes em serial (sem distribuição).

**1.2 - ARQUITETURA MESTRE-ESCRAVO**

A arquitetura consiste em vários “ambientes de processamento” (diferentes máquinas, processadores, …) chamados de escravos, onde são designadas tarefas por uma outra

**1.3 - CHECKSUM**

Checksum (soma de verificação) é um verificador de integridade de dados que passam por canais com ruído (que podem gerar perda de dado).

O algoritmo usado neste trabalho foi a checagem longitudinal de paridade, que consiste em quebrar um determinado dado em um conjunto Bytes e então computar um ou exclusivo (XOR) de cada Byte.  
 Para checar a integridade, o dado gerado pelo acumulo dos ou exclusivos de todos os dados do conjunto é adicionado no final do conjunto e a operação XOR é realizada mais uma vez. Caso o valor resultante seja diferente de zero então o dado não é consistente o que indica perda no canal ruidoso.

**1.4 - OBJETIVO**

O principal objetivo do trabalho foi realizar os checksums para diferentes tamanhos de vetores utilizando a arquitetura distribuída mestre-escravo em comparação com os mesmos feitos em serial.

Na distribuição os escravos se registram no mestre previamente. O mestre recebe um vetor enviado pelo cliente e o separa em partes iguais quando o vetor tem tamanho par e o distribui para seus escravos registrados para que realizem o checksum em seus subvetores e retornam o resultado, e quando o vetor é ímpar, todo o processo descrito anteriormente é repetido, porem um acréscimo do resto do tamanho do vetor original é adicionado ao último subvetor, então o mestre após receber todos os valores retornados realiza o checksum com os resultados e retorna o resultado para o cliente.

**5 - IMPLEMENTAÇÃO**

O trabalho é dado em Java usando RMI (Remote Method Invocation) e consiste em aplicar a arquitetura mestre-escravo para garantir processamento distribuído e paralelo.

Em nossa implementação apresentamos uma interface chamada Checksum, uma classe usada para garantir o funcionamento conciso em threads ChecksumCallable e as classes Master e Slave.

1. **Checksum:**utilizada pelo mestre e escravos, essa assinatura garante que tanto o mestre quanto os escravos tenham implementações para uma função de checagem de integridade e outras funções para cadastro e remoção. A interface estende RMI assim tornando-a uma interface capaz de ser remotamente invocada.
2. **ChecksumCallable:**A classe é filha de Callable, uma classe utilizada para comunicação entre threads.  
   ChecksumCallable é um receptáculo que armazena um elemento Checksum e uma lista de Bytes.   
   Este receptáculo garante que quando a thread for inicializada o método *call,* será chamado e então a lista contida no receptáculo será dada para o Checksum, também contido, resolver e depois retornará o resultado.
3. **Slave:**O escravo implementa a interface Checksum e também Serializable, garantindo que pode ser chamada remotamente.  
   Cada escravo possui um identificador que o identifica entre outros escravos.  
   O método *checksum* da interface Checksum faz a operação ou exclusiva bit a bit numa lista de Bytes e retorna o valor dessa operação.  
   O método *register* da interface Checksum invoca o método *register* do mestre e passa como argumento o próprio escravo para que o mesmo seja registrado no mestre.  
   O método *unregister* da interface Checksum mata o escravo.  
   A função *main* entra em contato com o mestre através do registrador de nomes provido pela RMI e invoca de imediato a função *register* garantindo que o escravo esteja disponível para o mestre.
4. **Master:**  
   O mestre implementa a interface Checksum.  
   O método *register* da interface Checksum registra um escravo encapsulando o tal em um objeto da classe ChecksumCallable (isso garante que cada escravo terá sua própria thread na hora de execução) e então armazenando-o em um hashmap.  
   O método *unregister* da interface Checksum remove o escravo.  
   O método *checksum* da interface Checksum invoca o método *futureMap* da própria classe Master, iniciando cada uma das threads contendo os escravos e passando para cada um pedaço da lista de Bytes para que possam ser processadas em paralelo. Logo após isso o *checksum* junta a resposta de cada escravo através de uma operação ou exclusiva em cima de cada retorno das threads e realiza também tratamento caso algum escravo seja morto no meio do processo. Por último, *checksum* retorna o valor acumulado dos *checksums* de cada escravo

Por último mas não menos importante, temos a classe Client, onde todos os testes São realizados

**6 - MANUAL DE EXECUÇÃO**

**Interface Checksum.java**

****

**ChecksumCallable.java**

****

**Master.java**

****

**Slave.java**

****

Para que se possa executar o Sistema é necessário que o Java esteja instalado corretamente.

1. Entrar no diretório desejado, neste caso “~/TrabalhoPP1”
2. Gerar os .class desejados
   1. Mestre:
   2. Escravo:
   3. Cliente:
3. Depois dos .class gerados podemos executar o Sistema na seguinte ordem
   1. Mestre:
   2. Escravo(s):
   3. Cliente:

**7 - ANÁLISE DO DESEMPENHO**

**8 - CONCLUSÃO**