Exercício 4 - Análise de Threads em `MultipleThreadExample.java`

Questão

O que acontece se na linha de comando digitada via console `java MultipleThreadExample.java N`, o valor de N for aumentando? Há um limite pragmático para N? Justifique!

Resposta

O que acontece quando N aumenta?

Quando o valor de **N** (número de threads) é aumentado no programa `MultipleThreadExample.java`, observamos os seguintes comportamentos:

1. Divisão do Trabalho

O programa divide a tarefa de gerar **1 bilhão de números aleatórios** entre N threads:

```
final long numeroPorThread = TOTAL_NUMEROS / threads;
```

- Com **N = 1**: 1 thread processa 1.000.000.000 números
- Com **N = 2**: cada thread processa 500.000.000 números
- Com **N = 4**: cada thread processa 250.000.000 números
- Com **N = 100**: cada thread processa 10.000.000 números

2. Ganho de Performance Inicial

- Para valores pequenos de N (até o número de cores da CPU), há **ganho significativo** de performance devido ao paralelismo real
- Cada thread executa simultaneamente em um núcleo diferente do processador
- O tempo total de execução **diminui** proporcionalmente ao número de cores disponíveis

3. Degradação de Performance após Certo Ponto

Quando N excede o número de núcleos físicos disponíveis:

- O sistema operacional precisa fazer **context switching** (alternância de contexto) entre threads
- O **overhead de criação e gerenciamento** de threads aumenta
- Múltiplas threads competem pelos mesmos recursos (CPU, cache, memória)
- O tempo total pode **aumentar** ao invés de diminuir

Limites Pragmáticos para N

1. **Limite Lógico**

Como o programa gera exatamente **1 bilhão de números**, existe um limite lógico superior:

$$N_{max_logico} = 1.000.000.000$$

Teoricamente, você poderia criar até 1 bilhão de threads, onde cada thread geraria apenas 1 número. Porém, isso é **completamente impraticável**.

2. **Limite de Hardware (Núcleos da CPU)**

O **ponto ótimo** para N geralmente é:

$$N_{otimo} \approx \text{número de núcleos lógicos da CPU}$$

Exemplo:

- CPU com 4 cores / 8 threads: **N ótimo ≈ 8**
- CPU com 8 cores / 16 threads: **N ótimo ≈ 16**
- CPU com 16 cores / 32 threads: **N ótimo ≈ 32**

Acima desse valor, o ganho de performance é marginal ou até negativo.

3. **Limite de Memória**

Cada thread consome memória:

- **Stack space**: ~1 MB por thread (configurável com `-Xss`)
- **Objetos**: `Random`, variáveis locais, contexto de execução

Cálculo aproximado:

- 1000 threads ≈ 1 GB de memória apenas para stacks
- 10.000 threads ≈ 10 GB de memória

O sistema operacional pode **limitar o número máximo de threads** por processo.

4. **Limite do Sistema Operacional**

- **Linux**: limite configurável (geralmente 32.768 threads por processo)
- **Windows**: limite teórico alto, mas problemas práticos aparecem antes
- O JVM também pode impor limites internos

5. **Limite de Overhead**

Criar e sincronizar threads tem custo:

- **Criação de thread**: ~1-2 ms
- **Context switching**: microsegundos por troca
- **Sincronização**: contenção de recursos compartilhados

Exemplo prático:

- Para N = 10.000 threads:
  - Tempo de criação: ~10-20 segundos
  - Cada thread processa apenas 100.000 números
  - O overhead supera o benefício do paralelismo

Conclusão

Resposta Pragmática

Sim, existe um limite pragmático para N, que é determinado por **múltiplos fatores**:

Fator	Limite Prático
Ótimo de Performance	N = número de cores lógicos da CPU (4-32 tipicamente)
Limite de Memória	N < 1.000-10.000 (dependendo da RAM disponível)
Limite do SO	N < 32.768 (Linux) ou similar no Windows
Limite Lógico	N ≤ 1.000.000.000 (total de operações)

Recomendação

Para este programa específico, o **valor ideal de N** está entre:

$$4 \leq N \leq 32$$

Esse intervalo proporciona:

- Paralelismo efetivo
- Baixo overhead de criação
- Utilização otimizada dos recursos
- Tempo de execução mínimo

Justificativa Final

Aumentar N além do número de cores não traz benefícios porque:

1. O processador não consegue executar mais threads simultaneamente
2. O sistema operacional precisa fazer context switching constante
3. O overhead de gerenciamento supera o ganho de paralelismo
4. A contenção por recursos (cache L1/L2/L3, memória) aumenta

Fórmula empírica para N ideal:

$$N_{ideal} = \text{cores físicos} \times (1 + \frac{\text{tempo_bloqueio}}{\text{tempo_CPU}})$$

Como este programa é **CPU-bound** (sem I/O ou bloqueios):

$$N_{ideal} \approx \text{número de cores lógicos}$$

Teste Prático Sugerido

Execute o programa com diferentes valores de N e meça o tempo:

```
# Compilar
javac MultipleThreadExample.java

# Testar com diferentes valores
java Ex3.MultipleThreadExample 1
java Ex3.MultipleThreadExample 2
java Ex3.MultipleThreadExample 4
java Ex3.MultipleThreadExample 8
java Ex3.MultipleThreadExample 16
java Ex3.MultipleThreadExample 32
java Ex3.MultipleThreadExample 64
java Ex3.MultipleThreadExample 128
```

Observação esperada:

- Tempo **diminui** de 1 até ~número de cores
- Tempo **estabiliza** próximo ao número de cores
- Tempo pode **aumentar** para valores muito altos (overhead)

Documento elaborado para ECM251 - Aula 22