RELATÓRIO: ACERCA DA APLICAÇÃO DE VERSÕES DE THREADS PARA EXCUÇÃO DE UM MESMO PROCESSO

Aluno(a): Larissa Carlos de Paulo

Professor: João Robson Santos Martins

Brasília – DF

2024

THREADS

Em Java as threads são como pequenos processos que podem atuar simultaneamente dentro de um programa. Elas possibilitam diferentes partes do código possam ser executadas ao mesmo tempo, melhorando a eficiência e a performance do programa, assim threads podem ser definidas como:

> Processo com entidade própria, com próprio contexto de escalonamento, mas que compartilha a estrutura de dados com seu pai. *Thread* ou processo leve é uma unidade básica de utilização de CPU que consiste em: apontador de instruções, conjunto de registradores e espaço de pilhas. (SCHEFFER, 2007)

A partir do exposto acima, threads são processos leves que, mesmo tendo suas próprias entidades e contextos de escalonamentos, compartilham a estrutura de dados com o processo pai. Devido a esse compartilhamento é possível uma comunicação e troca de dados mais eficiente entre diferentes partes do programa.

THREADS E SEU FUNCIONAMENTO COMPUTACIONALMENTE

A atuação computacional das threads permite que estes operam de maneira semelhante aos processos tradicionais, especialmente em termos de estados. Elas podem estar em estados de pronto, bloqueado, executando ou terminado. No entanto, ao contrário dos processos mais pesados, apenas uma thread de cada vez pode estar em execução na CPU, com a alternância ocorrendo rapidamente para simular a simultaneidade.

Cada thread irá operar sequencialmente dentro de seu contexto, mas devido à natureza leve das threads, o gerenciamento e a troca entre elas são mais eficientes e consumindo menos recursos do que processos completos.

> As threads operam de forma semelhante a processos quanto ao seu estado, podem ser pronto, bloqueado, executando e terminado, apenas uma thread de cada vez em execução na CPU, executa seqüencialmente e pode criar threads filhas. (SCHEFFER, 2007)

Além disso, as threads têm a capacidade de criar threads filhas, possibilitando expandir ainda mais a capacidade de paralelismo dentro de um programa. As threads podem ser um recurso particularmente útil para dividir tarefas complexas em tarefas menores e independentes, menos complexas, que podem ser executadas simultaneamente, melhorando a eficiência geral do processamento.

Portanto, mesmo com a restrição de uma única thread em execução por CPU a qualquer momento, o uso de múltiplas threads e sua rápida alternância proporcionam uma performance otimizada e uma melhor utilização dos recursos do sistema.

USO DE THREADS E SEU IMPACTO NA EXECUÇÃO DE UM ALGORITMO

O uso de threads pode afetar significativamente o tempo de execução de um algoritmo ao permitir a exploração do paralelismo real. Em sistemas com múltiplos processadores, várias threads podem ser executadas simultaneamente em diferentes núcleos, e isso acelera a execução de tarefas dividindo o trabalho entre os processadores disponíveis. Esse paralelismo aumenta o número de atividades executadas por unidade de tempo, o que leva a uma conclusão mais rápida do algoritmo em comparação com uma execução sequencial.

> Permitir a exploração do paralelismo real oferecido por máquinas multiprocessadores. Aumentar número de atividades executadas por unidade de tempo (throughput). Aumentar tempo de resposta, possibilidade de associar threads a dispositivos de entrada/ saída. Sobrepor operações de cálculo com operações de entrada e saída. (SCHEFFER, 2007)

Além disso, o uso de threads pode aumentar o tempo de resposta e a eficiência geral do sistema. Associar threads a dispositivos de entrada e saída permite que operações de cálculo sejam sobrepostas com operações de I/O.

Dessa forma, enquanto uma thread aguarda a conclusão de uma operação de entrada ou saída, outras threads podem continuar executando. Ao realizar essa sobreposição, ocorre a redução do tempo ocioso do processador e melhora a utilização dos recursos do sistema, resultando em um desempenho mais ágil e responsivo do algoritmo.

RELAÇÃO ENTRE OS MODELOS DE COMPUTAÇÃO - CONCORRENTE E PARALELO E SUA PERFORMANCE EM ALGORITMOS.

Os modelos de computação concorrente e paralela e a performance dos algoritmos são fundamentais para otimizar a execução de tarefas complexas. Percebeu-se que a distribuição das barreiras é um fator determinante no desempenho paralelo.

Quando as barreiras, que são pontos de sincronização entre threads, são distribuídas de forma igualitária, cada thread pode progredir de forma usual sem ser desnecessariamente bloqueada, resultando em um fluxo de trabalho mais eficiente. Garantindo que todas as threads compartilhem a carga de trabalho de maneira equilibrada, evitando os gargalos e melhorando o sistema:

> Com base nos estudos de caso, percebe-se que a distribuição das barreiras é um fator determinante no desempenho paralelo. O ganho de desempenho foi superior ao ideal nos casos em que a distribuição das barreiras era feita de forma igualitária entre as threads. (DA SILVA SERPA,2013)

O ganho de desempenho é superior ao ideal nos casos em que a distribuição das barreiras era feita de forma igualitária entre as threads. Isso ocorre porque uma distribuição equilibrada minimiza o tempo de espera e maximiza a utilização dos recursos da CPU, permitindo que o algoritmo se beneficie plenamente do paralelismo oferecido pelo hardware.

Em contrapartida, uma distribuição desigual pode levar a situações em que algumas threads ficam ociosas aguardando outras, resultando em subutilização/precarização dos recursos levando a um aumento do tempo total de execução.

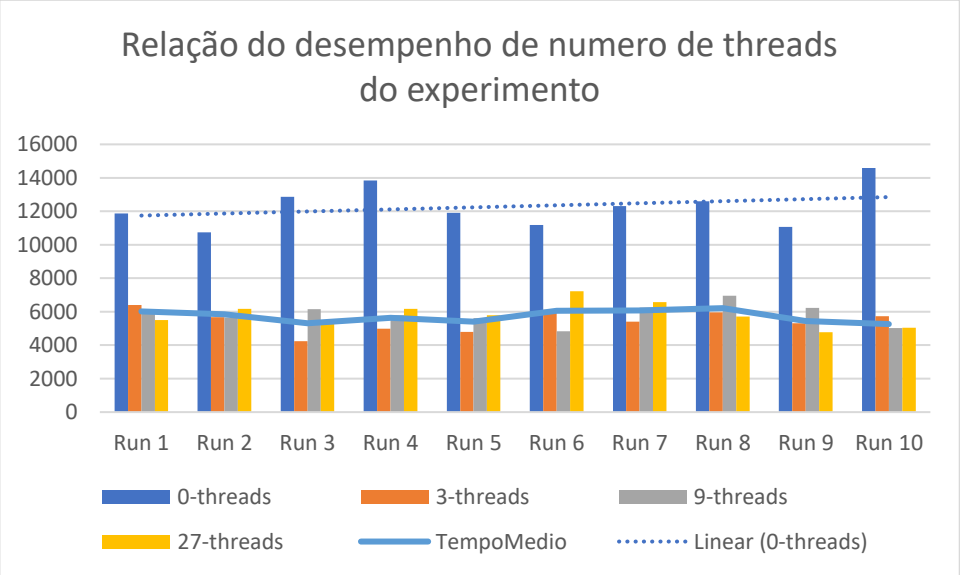
Portanto, com a implementação eficaz de modelos de computação concorrente e paralela, e uma distribuição eficiente das barreiras, o ganho de performance nos algoritmos será verificado.

ANALISANDO OS RESULTADOS OBTIDOS

Comparison of Execution Times (Each run in milliseconds)

Experiment	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average
Reference (0 threads)	11864	10734	12859	13849	11908	11188	12317	12602	11061	14595	12297
3 threads (each 9 cities)	6396	5683	4243	4982	4802	6109	5406	5965	5305	5727	5461
9 threads (each 3 cities)	6159	5663	6158	5782	5610	4833	6247	6961	6227	5019	5865
27 threads (each 1 city)	5498	6176	5501	6167	5781	7215	6581	5716	4777	5038	5845

O print acima está relacionado a resposta adquirida pelo programa, para aferição dos dados, após os teste de desenvolvimento foi necessário utilizar o internet móvel para coleta da última remessa de dados apresentada, dessa forma o tempo para execução foi afetado pela conexão móvel, no entanto como será demonstrado, o resultados se mantiveram proporcionais. Assim a análise dos gráficos seguirá normalmente:



- No eixo X, apresenta a quantidade de vezes que o experimento foi executado.
- No eixo Y, indicaremos o tempo médio de execução.
- Cada barra do gráfico representa uma versão do programa (0, 3, 9 e 27 threads) respectivamente.
- Vamos normalizar os tempos de execução dividindo-os pelo tempo obtido com apenas uma thread (0-threads), como exposto na linha de tendência.

A linha de tempo-médio:

- Mostra as proporções de tempo em relação à execução com uma única thread. O gráfico mostra que o tempo médio de execução diminui à medida que aumentamos o número de threads.
- Após um certo ponto (por volta de 9 threads), os ganhos de desempenho diminuem.
- Isso pode ser devido sobrecarga de sincronização entre as threads ou limitações do hardware (internet móvel).

A linha de tendencia:

- A linha de tendência permite comparar as proporções de tempo em relação à execução com uma única thread.
- Observamos que, inicialmente, há uma melhoria significativa à medida que adicionamos threads.
- No entanto, essa melhoria diminui à medida que aumentamos o número de threads.

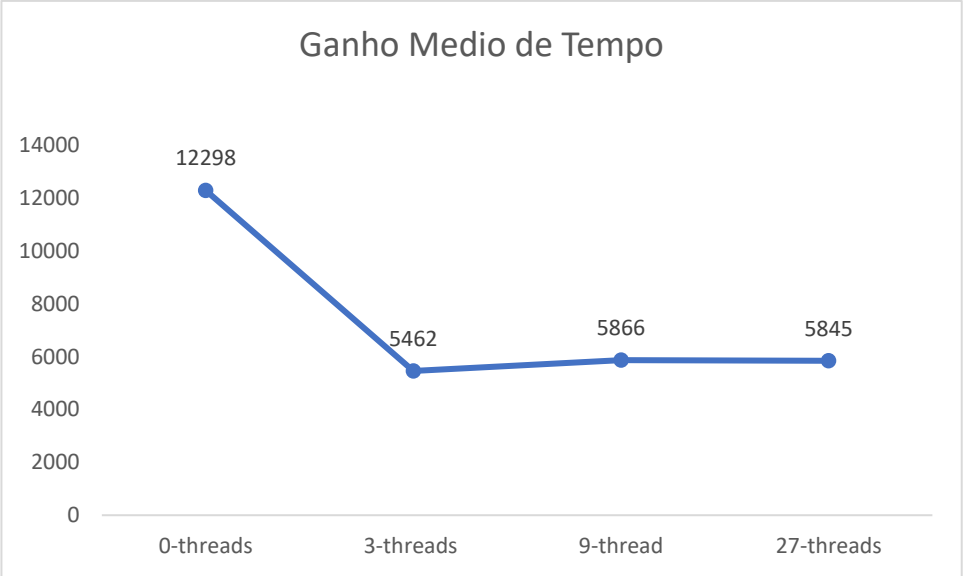
Análise dos dados:

Foi observado que o tempo médio de execução varia com o aumento do número de threads. Também se percebeu que há ganhos significativos de desempenho à medida que adicionamos mais threads, no entanto ao elevar a quantidade para números maiores que 3 o ganho é pouco significativo, assim tendo apresentado a versão com 3- threads um desempenho mais equilibrado, ou seja:

Adicionar threads pode melhorar o desempenho, mas há um ponto de retorno decrescente.

Faz-se necessário equilibrar o número de threads e a sobrecarga de sincronização.

O gráfico seguinte deverá abordar o comparativo e ganho médio de tempo após as 10 execuções:



- No eixo X, apresenta a quantidade de threads do experimento (0, 3, 9 e 27).
- No eixo Y, indicaremos o ganho médio de execução.
- Assim como no anterior o tempo médio diminuiu conforme a adição de threads na execução do programa.
- Após a 9-threads, o programa parece estar estabilizado, de forma que o número ideal aparenta estar entre 3 e 9 threads.

Análise dos dados:

A queda inicial no tempo médio de execução indica que a paralelização (usando várias threads) está melhorando o desempenho. No entanto, a partir de 9 threads, a diminuição no tempo médio é menos acentuada.

BIBLIOGRAFIA

DA SILVA SERPA, Matheus et al. Avaliando Diferentes Interfaces de Programação Paralela em Simulação de Fluxos de Fluidos.

SCHEFFER, Rosely. Uma visao geral sobre threads. **Revista Campo Digital**, v. 2, n. 1, 2007.