Universidade Federal do Rio Grande do Norte

Instituto Metrópole Digital

Análise e Comparação de Speedup e Eficiência do código serial e paralelo na geração de um histograma com atualizações em tempo de execução

João Vitor Venceslau Coelho

Natal/RN
2020

Introdução

Este relatório consiste na explicação de como **gerar um histograma com atualizações em tempo de execução dado um vetor de números**, buscando deixar claros todos os detalhes do objetivo a ser cumprido e da abordagem utilizada, sendo que esta usa uma distribuição normal para gerar os números a serem processados à geração do histograma. As versões serial e paralela do algoritmo implementado serão explanadas, em seguida os resultados obtidos serão mostrados e comentados. Será também brevemente comentada a corretude dos algoritmos implementados e serão expostas as análises do speedup, da eficiência e da escalabilidade do algoritmo paralelo para o serial. Ao final, um resumo das atividades e análises realizadas estará apresentado, sintetizando os principais pontos do relatório.

Geração de um histograma com atualizações em tempo de execução

A geração de um histograma com atualizações em tempo de execução se caracteriza por durante o processamento dos números ocorrer a atualização parcial do histograma, assim este é construído aos poucos durante o processamento dos números. Dependendo da implementação utilizada a frequência de atualização pode variar, ocorrendo a cada número processado ou a cada **x** números processados, sendo uma escolha do idealizador do algoritmo. Salientando que, quando a atualização do histograma ocorre a cada **x** números é comumente chamada de atualização por **batch**.

Desenvolvimento

Abaixo são explicadas as versões serial e paralela do algoritmo de **geração de um histograma com atualizações em tempo de execução**. A parte do código relativa à contagem do tempo não está sendo indicada, mas o tempo contabilizado consiste no tempo gasto para: calcular os intervalos a serem utilizados na delimitação dos **bins** do histograma, sortear um número da distribuição normal implementada, processar o número para identificar a qual **bin** ele pertence e atualizar o array. Na versão paralela a contagem do tempo consiste em: criar e finalizar as threads necessárias, alocar o espaço para armazenar a contagem parcial de cada thread, alocar os números gerados por cada thread de acordo com o **batch** utilizado, sortear os números da distribuição normal, processar em qual intervalo o número sorteado pertence, realizar o lock e unlock do devido mutex e atualizar o histograma ao fim de cada **batch**, assim como gerar mais um **batch** de números da distribuição normal e, por último, liberar a memória alocada pela thread;

Para gerar os números da distribuição normal foi utilizada uma implementação do método de ziggurat realizada por John Burkardt, disponível neste link.

Abordagens implementadas

Algoritmo Serial da geração de um histograma com atualizações em tempo de execução

O algoritmo recebe como entrada os seguintes argumentos: uma **flag** que indica se os valores do histograma gerado devem ser exibidos ao fim do algoritmo, a **seed** utilizada na geração dos números, a **quantidade** de números a serem sorteados, o número **mínimo de intervalos** e o número **máximo de intervalos**, o **range** dos números, isto é, o quanto os valores sorteados podem se distanciar do valor central da distribuição e, por fim, o valor **central** da distribuição. Após inicializar a **seed** e os parâmetros para geração dos números da distribuição normal, é sorteada uma quantidade de **bins** entre os números **mínimos** e **máximos** indicados por parâmetro, após esse sorteio é calculado o estado inicial da geração dos números da distribuição, incrementando em uma unidade os valores da **seed**, do **range** e do valor **central** da distribuição e multiplicando esses três números, é obtido estado inicial: $estadoInicial = (seed + 1) \cdot (range + 1) \cdot (central + 1)$. Inicia-se então o processo de geração do histograma.

Com base no valor **central** e no **range** são delimitados os valores mínimos e máximos dos números gerados e com a divisão da diferença entre eles pela quantidade de **bins** sorteada é determinada a quantidade de valores que cada **bin** engloba: ($distance = \frac{(max - min)}{bins}$), assim, calcula-se os limites de cada **bin** a partir do valor min, somando uma certa quantidade de $distances$ com esse valor, sendo que o limite do último **bin** é o valor max.

Após terem sido calculados os devidos limites inicia-se a geração dos números, seguindo a distribuição normal, para cada número gerado é feita uma verificação para encontrar a qual **bin** ele pertence, isso é realizado comparando o número com cada um dos intervalos definidos pelos limites calculados anteriormente. Ao identificar o intervalo correto, incrementa-se o contador do histograma relativo ao **bin** associado ao intervalo. Após sortear e classificar no **bin** correto todos os números, temos o histograma completo e os limites utilizados para os intervalos dele. Caso a **flag** seja maior que 0, é exibido o histograma, isto é, a quantidade de números em cada **bin**. Por fim ocorre a liberação da memória usada para armazenar os **bins** e os intervalos que os delimitavam.

Algoritmo Paralelo da geração de um histograma com atualizações em tempo de execução

A versão paralela do algoritmo possui como primeiro argumento o número de threads a serem usadas, os argumentos restantes são os mesmo utilizados na versão serial.

Os passos realizados são os mesmos até o calculo do estado inicial da geração de números da distribuição normal, após esse cálculo é feita a divisão da quantidade de números a serem sorteados pelo número de threads disponíveis, o resultado dessa divisão é o tamanho padrão de números a serem processados em cada thread. Em seguida, reserva-se a memória para armazenar os valores em cada **bin** do histograma e a memória para as threads e mutexes necessários. São então iniciados os mutexes de cada **bin** e o mutex para a geração dos números da distribuição normal. Após essa inicialização dos devidos mutexes, dá-se início ao processo de geração do histograma, assim como na versão serial, calcula-se os limites mínimos e máximos, aloca-se a memória para armazenar os valores deles e em seguida cria-se as threads para gerar o histograma. Cada thread realiza as seguintes operações:

Utilizando o seu rank e o tamanho padrão de números a processar, calcula o início e o fim dos números a sortear e processar. Se for a última thread a ser criada, então o fim dos números a utilizar é a própria quantidade solicitada por parâmetro, isso evita que a divisão entre a quantidade de números e a quantidade de threads tenha que ser exata.

Em seguida, é alocado o espaço para armazenamento local e parcial do números processados, o contador é iniciado, o tamanho do **batch** é determinado utilizando como base a quantidade padrão de números a serem processados e é alocado um array para armazenar os números a serem utilizados no **batch**.

Após realizar o lock do mutex, relativo a geração dos números, o array alocado previamente para os números a serem utilizados no processamento é preenchido, após o término desse preenchimento o mutex é liberado, isso ocorre para garantir que os mesmos números serão utilizados para gerar o histograma, pois sem esse cuidado, é provável que a toda vez que o programa for executado um histograma diferente seja gerado.

Essas foram as preparações para dar início ao processamento dos números, assim, no início de cada laço um número do array é resgatado e devidamente alocado a um **bin** do histograma parcial e o contador é incrementado, quando o valor do contador é igual ao tamanho do **batch**, ocorre a atualização do histograma em construção, onde para atualizar cada **bin** dele o devido lock é chamado, permitindo assim que cada thread atualize um **bin** do histograma em desenvolvimento de forma paralela, claro, por se tratar de uma distribuição normal, onde a maior parte dos valores se concentram no centro, a maioria das threads sempre irão solicitar os mesmos **bins** centrais, causando o travamento das threads no momento de acesso aos devidos locks, entretanto nos momentos em que **bins** longe do centro da distribuição forem solicitados, esse mecanismo, de haver um lock por **bin,** será útil, pois permite que mais de uma thread atualize simultaneamente o histograma que está sendo elaborado. Atualizando o histograma, o valor do **bin** no histograma parcial é zerado, como o **batch** foi totalmente processado, a thread trava novamente o mutex de geração dos números para gerar um novo **batch** de números da distribuição e reseta o contador.

Após todos os números destinados a essa thread serem processados, são percorridos os **bins** locais da thread e se houver algum valor a ser adicionado ao histograma em

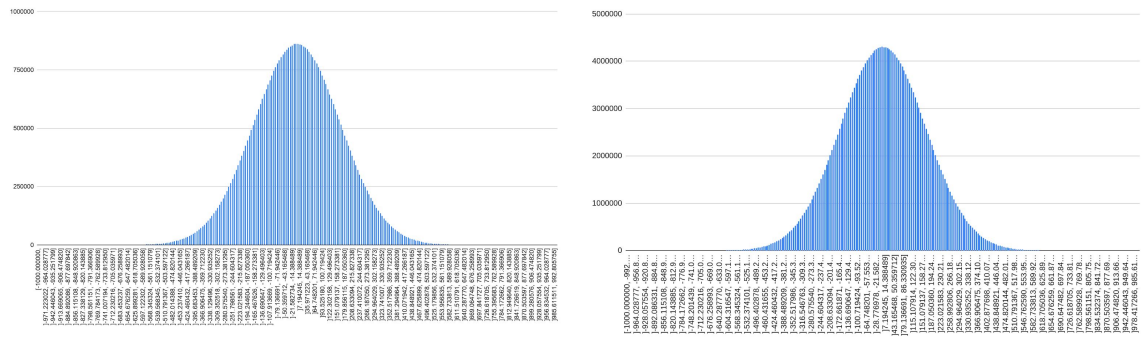
construção, o devido mutex é travado e então a atualização ocorre, para então liberar o mutex. Por fim, a memória alocada na thread é liberada, sendo elas: a memória para o histograma parcial e a para os números da distribuição usados em cada **batch.**

Após o término de todas as threads, temos o histograma final gerado e os devidos limites de seus **bins**. Caso a **flag** seja maior que 0, é exibido o histograma, isto é, a quantidade de números em cada **bin**. Por fim ocorre a liberação da memória usada para armazenar os **bins**, os intervalos que os delimitaram e as threads, ressaltando que os mutexes usados também são destruídos.

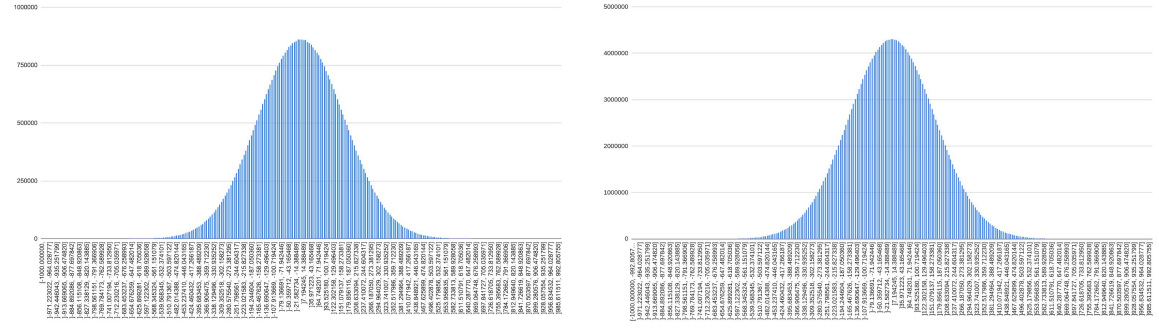
Resultados encontrados

Corretude do Algoritmo da geração de um histograma com atualizações em tempo de execução (Serial e Paralelo)

A corretude do algoritmo pode ser visualizada com os seguintes exemplos, onde pode-se perceber a distribuição dos números gerados:



Figuras 1 e 2: Apresentam a distribuição obtida no menor (esquerda) e maior (direita) quantidades de números utilizados no experimento com o algoritmo serial, 50000000 números e 250000000 números, respectivamente.



Figuras 3 e 4: Apresentam a distribuição obtida no menor (esquerda) e maior (direita) quantidades de números utilizados no experimento com o algoritmo paralelo, 50000000 números e 250000000 números, respectivamente.

Essas distribuições são obtidas ao executar os algoritmos com a **flag** ativa em 1, o programa irá escrever na saída um .csv com a frequência encontrada em cada intervalo.

Análise de Speedup, Eficiência e Escalabilidade

Abaixo está a tabela com as médias dos tempos válidos obtidos, sendo que para cada instância analisada, foram coletados 15 tempos, em que os dois maiores e os dois menores foram desconsiderados como válidos para o cálculo dessa média.

Geração de um histograma com atualizações em tempo de execução

Tamanho do Problema	Serial	4	8	16	32
50000000	32,7266670	6,8186620	3,4801240	2,6686810	2,3191530
100000000	65,4222220	13,6283370	6,9449720	5,0063960	4,2647490
150000000	98,0066670	20,4380010	10,5062460	7,4794830	6,1530290
200000000	130,6744440	27,2458520	13,7429020	9,9616080	8,3964140
250000000	163,3633330	34,0757120	17,1377140	11,6945220	10,2855060

Tabela 1: Apresenta os tempos médios obtidos nos experimentos por quantidade de números e algoritmo utilizados, assim como a quantidade de cores na versão paralela.

Percebe-se que o ganho de tempo com a versão paralela acaba diminuindo conforme o aumento do número de processadores, mesmo que o tempo gasto seja menor.

Adiante as tabelas apresentam os speedups e as eficiências para cada quantidade de processadores e de números, e em seguida temos os devidos gráficos.

Tamanho do Problema	Speedup 4	Speedup 8	Speedup 16	Speedup 32
50000000	4,799573	9,403879	12,263236	14,111477
100000000	4,800455	9,420084	13,067728	15,34023
150000000	4,795316	9,32842	13,103402	15,928197
200000000	4,796123	9,508505	13,117806	15,563125
250000000	4,794128	9,532388	13,969219	15,882867

Tabela 2: Apresenta o speedup da execução da versão paralela para a versão serial em cada tamanho de problema e quantidade de cores utilizada.

Tamanho do Problema	Eficiência 4	Eficiência 8	Eficiência 16	Eficiência 32
50000000	1,199893	1,175485	0,766452	0,440984
100000000	1,200114	1,177511	0,816733	0,479382
150000000	1,198829	1,166052	0,818963	0,497756
200000000	1,199031	1,188563	0,819863	0,486348
250000000	1,198532	1,191548	0,873076	0,49634

Tabela 3: Apresenta a eficiência da execução da versão paralela para a versão serial em cada tamanho de problema e quantidade de cores utilizada.

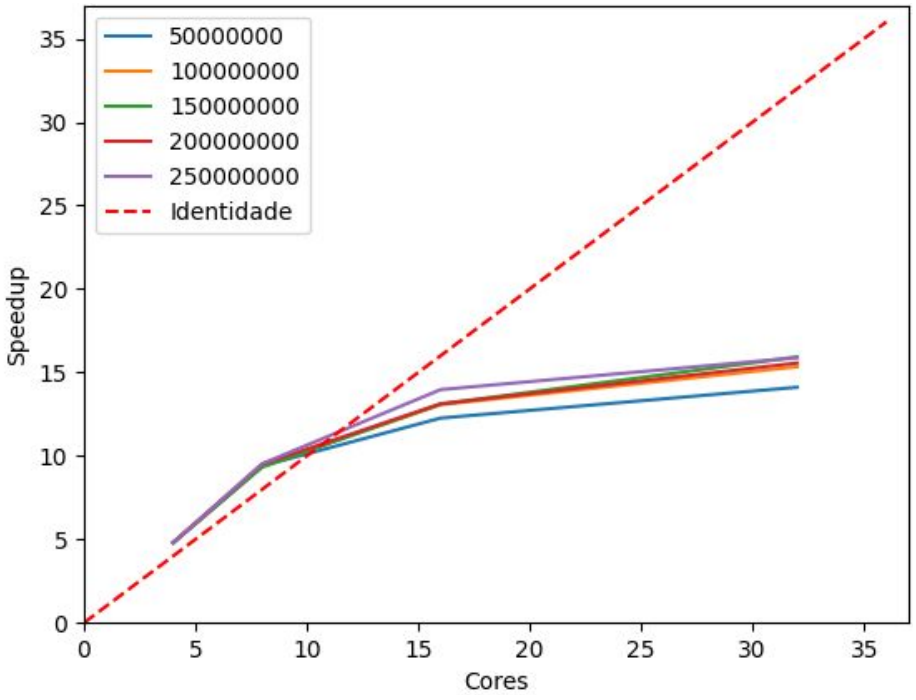
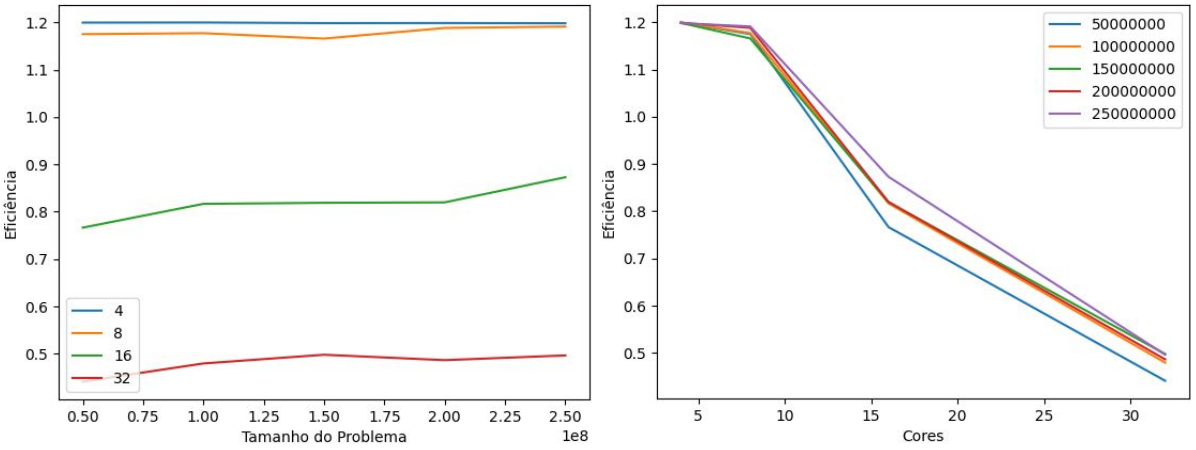


Figura 5: Comparação dos speedups obtidos em cada tamanho do problema com o speedup esperado.

Pode-se perceber que o speedup com 4 e 8 cores foi superlinear, porém conforme o número de cores aumenta para 16 e 32 o speedup diminui. Acredita-se que essa redução do speedup se dá pela existência das áreas críticas do algoritmo. À medida em que mais threads existem, maior a competição pelo acesso a essas áreas, o que acaba diminuindo o ganho final do tempo que se poderia obter.



Figuras 6 e 7: Nas comparações com a eficiência, pode-se perceber que ela se mantém constante, com leves indícios de crescimento em relação a quantidade de números, enquanto diminui de acordo com o aumento dos cores, sendo que esse decaimento é mais abrupto à partir da utilização de 16 cores. Até 8 cores a diminuição da eficiência era mais suave.

Diante desses dados, o algoritmo paralelo se categoriza como **Fracamente Escalável**, pois aumentando a quantidade de números e o número de cores em proporções iguais, a eficiência se mantém quase constante, com algumas pequenas flutuações crescentes.

Considerações Finais

Neste relatório apresentou-se brevemente como gerar um histograma com atualizações em tempo de execução. Foram explanadas as versões serial e paralela do mesmo algoritmo, sendo que a versão serial atualiza o histograma a cada elemento processado e a versão paralela atualiza esse mesmo histograma a cada x elementos processados, isso ocorre, porque uma região crítica existe, e é desejável a diminuição do acesso a ela, pois diminui a competição das threads pelo recurso.

Foi realizada uma breve demonstração da corretude dos algoritmos implementados, apresentando o resultado da execução dos mesmos por meio do gráfico de barras obtido com os **bins** do histograma gerado. Nos gráficos apresentados, foi facilmente perceptível a curva da distribuição normal utilizada na geração dos números usados para criação do histograma.

Foi feita a exposição das médias dos tempos obtidos, assim como os speedups e eficiências por meio de tabelas e os gráficos gerados por esses valores, os mesmo foram comentados e algumas pequenas justificativas para os comportamentos foram dadas.

Por fim, foi feita a categorização do algoritmo em relação a escalabilidade, concluindo que o algoritmo é Fracamente Escalável, dado o comportamento identificado na relação da eficiência com a quantidade de números usada.

Referências

Introdução a Sistemas Paralelos

Generate histogram in c

Calculate the histogram with OpenMP

Fill histograms (array reduction) in parallel with OpenMP without using a critical section