

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Processamento Paralelo e Distribuído – Turmas 01 e 02 (EARTE) – 2022/1

Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br

Laboratório I – Paralelismo de Processos e Threads

Objetivo:

Experimentar o paralelismo por meio de processos e *threads* na ordenação de um vetor de grandes dimensões. Comparar o tempo de execução da tarefa de ordenação de vetores com diferentes quantidades de processos ou *threads*.

Instruções:

Recentemente, você aprendeu sobre paralelismo de processos e *threads*, que permitem que um usuário configure e execute paralelamente várias instâncias de um único processo. Neste laboratório você programará um algoritmo básico de ordenação mergesort com múltiplos processos (ou *threads*).

Como exemplo, considere o vetor a seguir:

10	49	9	34	32	37	37	48	45	19	5	31	20	19	29	22	45	30	40	31	35	8	36	39	14
----	----	---	----	----	----	----	----	----	----	---	----	----	----	----	----	----	----	----	----	----	---	----	----	----

O seu programa de ordenação paralela receberá dois argumentos de linha de comando: o número de vezes que os dados devem ser divididos como parte da ordenação mergesort (*k*) e o tamanho do vetor de entrada a ser ordenado (*n*). Por exemplo, um *k* igual a 5 indica que os dados devem ser divididos em 5 segmentos de tamanho iguais, aproximadamente. Se considerarmos executar este exemplo com um *k* igual a 5, os dados serão divididos da seguinte maneira:

Split #1	10 49 9 34 32	Split #2	37 37 48 45 19	Split #3	5 31 20 19 29	Split #4	22 45 30 40 31	Split #5	35 8 36 39 14
----------	---------------	----------	----------------	----------	---------------	----------	----------------	----------	---------------

A primeira parte do mergesort paralelo exige que você classifique cada ordenação individual, em paralelo. Em nosso exemplo, isso significa que você lançará cinco *threads* (ou processos) – uma para cada uma das divisões – e cada processo (ou *thread*) deve classificar apenas seu segmento da entrada. Você não precisa escrever sua própria classificação aqui; na verdade, você pode usar bibliotecas prontas em C, Python ou Java, se preferir.

Depois que todos os processos (ou *threads*) de ordenação forem concluídos, os splits ficarão parecidos com:

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Split #1 9 10 32 34 49	Split #2 19 37 37 45 48	Split #3 5 19 20 29 31	Split #4 22 30 31 40 45	Split #5 8 14 35 36 39
---------------------------	----------------------------	---------------------------	----------------------------	---------------------------

Em seguida, o seu programa pegará pares adjacentes e os mesclará (merge), também em paralelo. Isso será feito em rodadas, onde a cada rodada um número de fusões (merge) paralelas ocorrem até que reste apenas uma única lista ordenada. Lembre-se, mesclar é uma operação $O(n)$ NÃO $O(n^*log(n))$ e por isso você **não deve** chamar usar bibliotecas de ordenação neste passo do algoritmo.

Em nosso exemplo, a primeira rodada consiste em duas fusões paralelas: Split #1 e Split #2; Split #3 e Split #4.

Split #(1+2) 9 10 19 32 34 37 37 45 48 49	Split #(3+4) 5 19 20 22 29 30 31 31 40 45	Split #5 8 14 35 36 39
--	--	---------------------------

A segunda rodada consiste em apenas um merge:

Split #(1+2+3+4) 5 9 10 19 19 20 22 29 30 31 31 32 34 37 37 40 45 45 48 49	Split #5 8 14 35 36 39
---	---------------------------

A última rodada também consiste em apenas um merge:

Split #(1+2+3+4+5) 5 8 9 10 14 19 19 20 22 29 30 31 31 32 34 35 36 37 37 39 40 45 45 48 49

Neste laboratório você deve sempre começar com a divisão “mais à esquerda” e mesclá-la com sua vizinha. Como tal, a divisão “mais à direita” geralmente será a última a ser mesclada. Para nos informar sobre o andamento será necessário imprimir, na tela, algumas informações sobre os merges e as rodadas.

Requisitos:

Divida a entrada em k segmentos de forma que cada segmento tenha o mesmo tamanho (por exemplo: 25 números de entrada / 5 segmentos = 5 números / segmento). Se você não conseguir obter segmentos de tamanho igual, deve garantir que os primeiros ($k - 1$) segmentos tenham o mesmo tamanho e que o último segmento seja menor que os demais em 1 unidade.

Inicie uma thread/processo por segmento para ordenar cada segmento usando algum algoritmo de ordenação (implementação própria ou biblioteca). Seu programa deve bloquear até que todos os threads/processos terminem de ordenar. Cada

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

thread/processo deve imprimir na tela uma linha de status indicando o número de elementos ordenados (tamanho do segmento).

Inicie várias rodadas para mesclar os segmentos. Cada rodada deve bloquear até que todas as threads/processos na rodada atual terminem antes de avançar. Os encadeamentos devem ser mesclados em uma ordem específica, onde o segmento no início do conjunto de dados é mesclado com o próximo segmento no conjunto de dados. Ou seja, o segmento[0] deve ser mesclado com o segmento[1]. Esse padrão deve seguir tal que segmento[i] seja mesclado com segmento[i + 1] para todos os valores pares de i. Se houver um número ímpar de segmentos, você não deve mesclar o segmento restante na rodada atual.

Após cada rodada, você terá $(ct/2)$ ou $((ct/2) + 1)$ segmentos restantes, onde ct é o número de segmentos que você teve na rodada anterior. Você deve continuar o processo de mesclagem até restar apenas 1 segmento. Na ordenação por mesclagem, a operação de mesclagem NÃO DEVE ser uma operação $O(n * \log(n))$. Desconsidere a existência de elementos duplicados na lista a ser ordenada. Em vez disso, pense em como você pode mesclar duas listas classificadas em tempo $O(n)!!$

Já que estamos usando paralelismo, devemos ver uma aceleração entre usar um único segmento (onde todo o trabalho é feito por uma única thread) e usar muitos segmentos. Isso deve ficar mais claro em casos de teste muito grandes.

1. O trabalho pode ser feito em grupos de 2 ou 3 alunos: não serão aceitos trabalhos individuais ou em grupos de mais de 3 alunos;
2. Os grupos poderão implementar os trabalhos usando qualquer uma dentre as três linguagens de programação: C, Java ou Python;
3. O paralelismo pode ser implementado por meio de processos ou *threads*, a critério do grupo, porém observem as limitações de cada biblioteca linguagem de programação (algumas não suportam corretamente, verifiquem a sua escolha);
4. Como resultado final cada grupo deverá executar o trabalho para pelo menos 3 diferentes tamanhos da lista (use valores grandes para n) e k=1 (sem paralelismo), 2, 4, 8, 16. Plote o resultado do tempo de execução nessas diferentes combinações de k e n.

Bom trabalho!



Universidade Federal
do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Bibliografia:

- [1] Python multiprocessing — Process-based parallelism
<https://docs.python.org/3/library/multiprocessing.html>
- [2] Python threading — Thread-based parallelism
<https://docs.python.org/3/library/threading.html>
- [3] POSIX thread (pthread) libraries
<https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html>
- [4] c_pthreads [Programação Concorrente]
http://cocic.cm.utfpr.edu.br/progconcorrente/doku.php?id=c_pthreads
- [5] Java Threads Tutorial
<https://www.edureka.co/blog/java-thread/>
- [6] ALGORITMOS CONCORRENTES E SUA IMPLEMENTAÇÃO
<https://www.prp.unicamp.br/pibic/congressos/xxcongresso/paineis/101918.pdf>