Eusebiu-Mihai Burtescu 335CB

Documentatie Tema1 APD

Sectiunea 1 – Feedback

Ce critica pot oferi este faptul ca am pierdut extrem de mult timp cu set-up ul. Adica: a trb sa ma chinui sa fac un makefile cu maven si toate lucrurile necesare. Am avut probleme si cu el, adica nu prea era functional la inceput.

Ce pot zice e ca tema asta a fost ceva care pare practic. Chiar am implementat ceva paralelizat care ar avea sens si in lumea reala.

Sectiunea 2 – Strategia de paralelizare

Paralelizarea in implementarea mea consta in impartirea task-urilor de parsare si prelucrare a datelor intre mai multe thread-uri. In thread-ul principal (main) am citit path-urile catre fisierele de articole si continutul fisierelor suplimentare, iar apoi am impartit lista de path-uri in intervale start-end si le-am transmis thread-urilor. Fiecare thread a parcurs si prelucrat articolele sale in liste locale. In aceste liste locale am scos duplicatele (pe care le aveam salvate intr-o lista shared, pentru a nu fi duplicate inter-thread). Pe articolele ramase am realizat o sortare si le-am dat merge in barrier action pentru printare. In thread, dupa sortarea lor, am calculat si creat toate informatiile relevante procesarii datelor si statisticilor la nivel local. Aceste date au fost adaugate in structurile shared direct, majoritatea fiind relativ rapide si fara sa coste prea multe resurse sau timp. O exceptie este lista de cuvinte care a trebuit sa fie sortata. Pentru marirea speed-up-ului am format si sortat liste locale, pe care le-am dat merge similar ca in cazul listei globale de articole. Thread ul asteapta in final la barrierWrite pentru sincronizarea scrierii in fisiere.

Am folosit doua bariere (CyclicBarrier):

1. **barrierRead** – sincronizeaza thread-urile astfel incat toate sa termine citirea si parsarea articolelor inainte de a trece mai departe.
2. **barrierWrite** – sincronizeaza thread-urile dupa prelucrarea datelor si adaugarea acestora in structurile comune. Aceasta bariera include un **barrier action**, executat o singura data de ultimul thread care ajunge la bariera, unde se realizeaza scrierea rezultatelor in fisiere.

Alte mecanisme de sincronizare folosite:

- **ConcurrentHashMap** – pentru datele comune intre thread-uri (ex.: authors). Ofera operatii atomice fara blocari manuale.
- **ConcurrentSkipListSet** – pentru stocarea informatiilor comune, pastrand ordinea si eliminand duplicatele.
- **AtomicInteger** – pentru numararea articolelor procesate, utilizat pentru calculul numarului de duplicate.

Eusebiu-Mihai Burtescu 335CB

Design-ul este corect si eficient deoarece paralelizeaza majoritatea task-urilor de parsare si procesare, lasand serializate doar citirea si scrierea. O paralelizare completa a citirii si scrierii ar necesita utilizarea de lock-uri, ceea ce ar fi mult mai costisitor si ar reduce considerabil performanta.

Sectiunea 3 – Analiza de performanta si scalabilitate

Setup de testare:

OS: Linux Mint

CPU:AMD Ryzen 7 6800H

RAM: 32GB

JAVA: 25

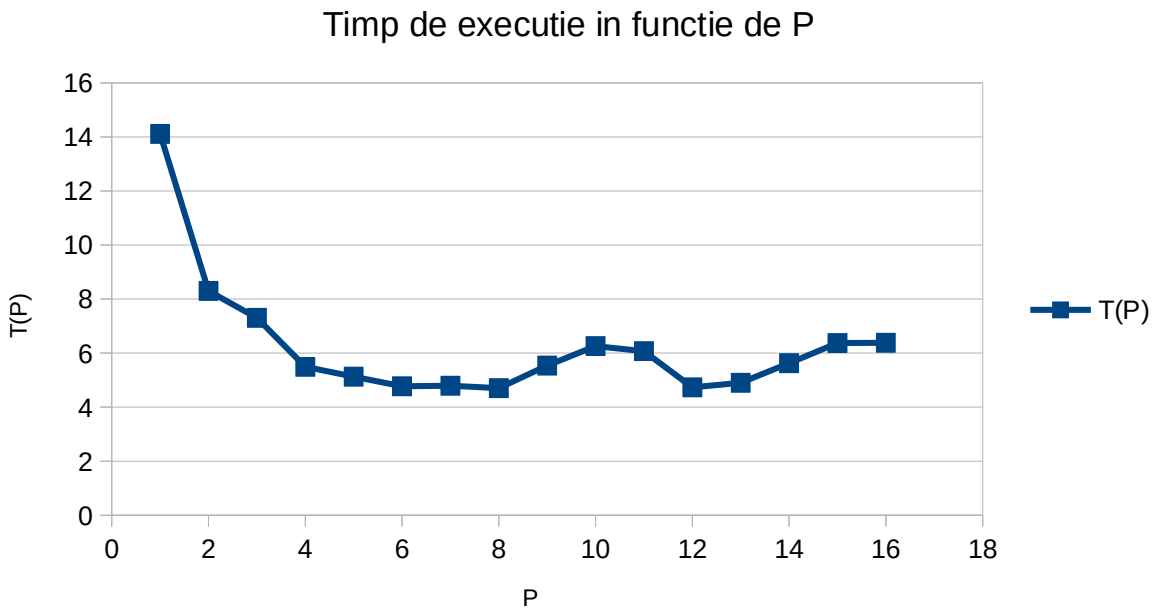
Nr Cores\Threads: 8\16

Dimensiune Data Sheet: Test_5 Checker

Tabel Date:

P	T(P)	S(P)	E(P)
1	14.107	1	1
2	8.301	1.699433803	0.849716902
3	7.302	1.931936456	0.643978819
4	5.49	2.569581056	0.642395264
5	5.126	2.752048381	0.550409676
6	4.77	2.957442348	0.492907058
7	4.79	2.945093946	0.420727707
8	4.7	3.001489362	0.37518617
9	5.537	2.54776955	0.283085506
10	6.26	2.253514377	0.225351438
11	6.07	2.324052718	0.21127752
12	4.731	2.981822025	0.248485169
13	4.9	2.878979592	0.221459969
14	5.624	2.508357041	0.17916836
15	6.372	2.213904583	0.147593639
16	6.38	2.211128527	0.138195533

Eusebiu-Mihai Burtescu 335CB



Observatie: Toate valorile T(P) sunt rezultatul mediei intre 3 rulari ale programului cu nr de threaduri P.

Incepand cu p=2, timpul scade seminifcativ pana al 4 -5 unde gain-ul de timp devine din ce in ce mai mic.

Se observa ca timpul se “stabilizeaza” de la p = 6. High urile si low uirle de urmeaza se datoreaza prezentei altor procese pe cores de care se ocupa CPU ul. Numarul de threaduri optim este 6 deoarece se observa te grafic cum are cea mai mica valoare a timpului de executie comparativ cu vecinii. Pentru p = 12 se vede un alt low dar nu este destul de bun pentru costul resurselor (12 threaduri)

Posibile cauze de incetinire:

-Asteptarea threadurilor la cele 2 bariere

-Numar mare de intrari, cauzand pierderi mari de timp in cazul parcurgerilor

-Prezenta operatiilor pe strucrurile speciale thread-safe care au mecanisme locale de sincornizare, nepermitand mai multor threaduri sa el acceseze simultan.