Zadanie zaliczeniowe Collatz, Mateusz Sulimowicz, ms429603

Wprowadzenie

UWAGA: Rozwiązanie nie zawiera implementacji zespołów procesów.

Podczas implementacji próbowałem różnych rzeczy:

- W zespołach współdzielących wyniki, jeśli po skończeniu obliczenia wątek zapisywał wyniki dla wszystkich wartości ciągu, które napotkał podczas obliczenia, to czasy były znacznie gorsze niż gdy zapisywany był tylko wynik dla wejścia calcCollatz(n). Wynika to z faktu, że wątek potrzebuje wyłączny dostęp do współdzielonej struktury żeby zapisać wynik. Nieważne czy wszystkie wyniki zapisywałby na raz, czy przy jednym dostępie do struktury zapisywałby jeden wynik - może spędzić dużo czasu w strukturze mając ją na wyłączność, co spowalnia pracę pozostałych wątków. Ponadto, nie mamy gwarancji, że wyniki częściowe w ogóle przydadzą się innym wątkom. Takie rozwiązanie może potrzebować dużo pamięci.

- wątek mógłby zapisywać do współdzielonej struktury na przykład co K-ty wynik, gdzie K jest jakąś dobraną stałą. Zmniejszyło by to liczbę zapisów w strukturze, ale skutkowałoby tym, że wątek i tak musiałby drugi raz przejść się po wszystkich napotkanych podczas obliczenia wartościach ciągu.

- W TeamAsync budowanie logarytmicznej ścieżki krytycznej (wywołanie obliczenia calcCollatz() dla i-tego wejścia powoduje wywołanie obliczenia dla $(2 * i)$-tego oraz $(2 * i + 1)$-wszego wejścia, a po zakończeniu własnego obliczenia czekanie na wywołane obliczenia) nie dawało obiecujących rezultatów. Często lepsze wyniki dawało wywołanie wszystkich obliczeń po kolei i po kolei czekanie aż się zakończą. Pojedyńcze wywołanie calcCollatz(n) może wykonywać się znacznie dłużej niż czas potrzebny na zebranie wyników.

Analiza

W zespołach -X zapisywane są do SharedResults tylko pary

$$(n, calcCollatz(n)),\ n \in contestInput.$$

W i-tej iteracji wątek sprawdza, czy obliczenie dla wejścia będącego i-tym elementem ciągu nie zostało już wykonane.

Przeanalizujmy wyniki konkursów dla contestId = 23. Lista *contestInput* składa się dla takich konkursów z:

- w konkursie SameNumber z samych liczb 24, rozmiar = 1000

- w konkursie ShortNumber z dużej liczby niewielkich liczb rzędu [24, .., 2400],

- w konkursie LongNumber z liczb, które mają od 24 do 48 cyfr. 24 liczby

Niezależnie od rozmiaru zespołu możemy zauważyć, że w konkursie:

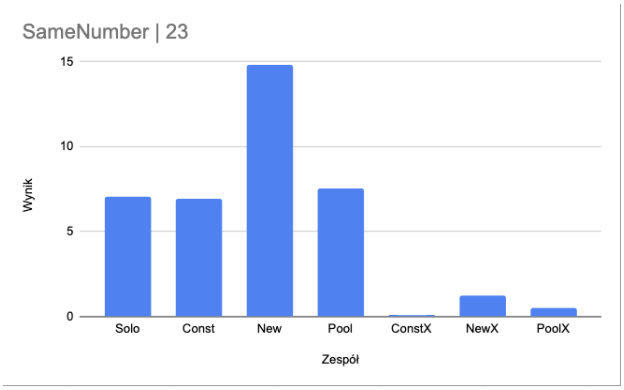
- SameNumber: Wielokrotnie obliczamy wynik dla tego samego wejścia, więc oczywiście czas wykonania przy współdzieleniu wyników częściowych jest bardzo krótki.

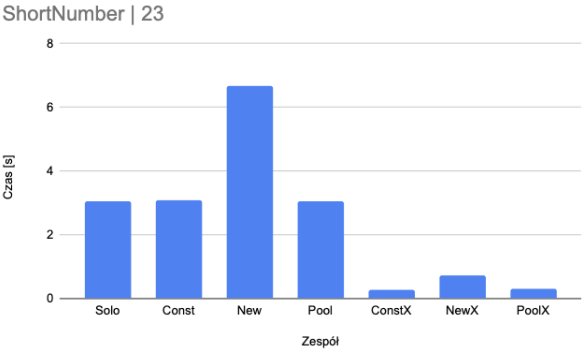
- ShortNumber: Obliczamy wyniki dla małych liczb. Zajmuje to jak widać niedużo czasu. Ponieważ wejścia są małe, jest duża szansa, że wyniki częściowe obliczenia przydadzą się w innych obliczeniach, więc czasy działania zespołów X są znacznie mniejsze od pozostałych.

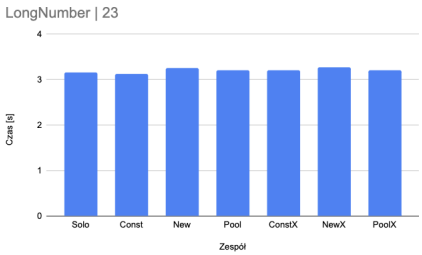
- LongNumber: Ponieważ obliczamy wyniki dla dużych liczb, to jest małe prawdopodobieństwo, że wynik obliczenia przyda się w innych obliczeniach.

1) Team.size = 1

- SameNumber: Obliczenie calcCollatz(24) zajmuje bardzo mało czasu. Z tego powodu większość czasu w zespole TeamNewThreads jest poświęcana na tworzenie nowych wątków, co wyróżnia się na wykresach. TeamConstX wygrywa, bo efektywnie liczy calcCollatz(24) tylko raz.

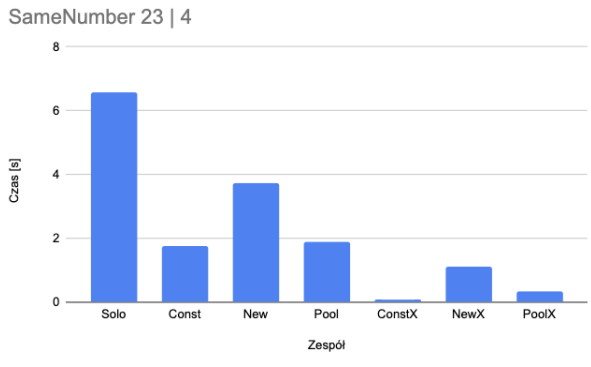


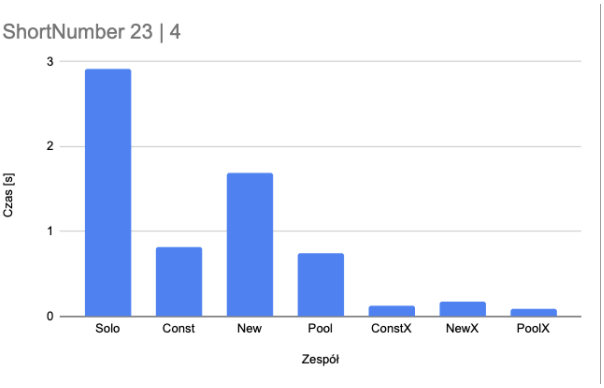


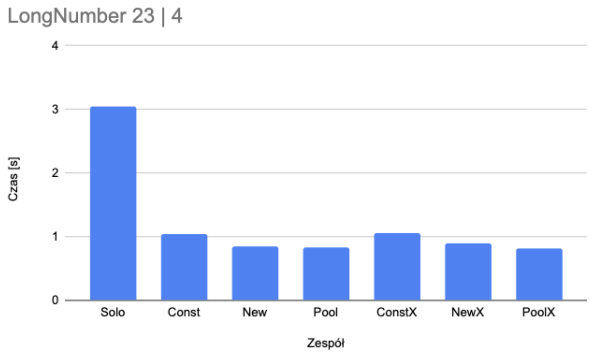


2) Team.size = 4

Ponieważ rozmiar zespołu jest już większy, to TeamNew nie traci aż tak dużo czasu na tworzeniu nowych wątków.





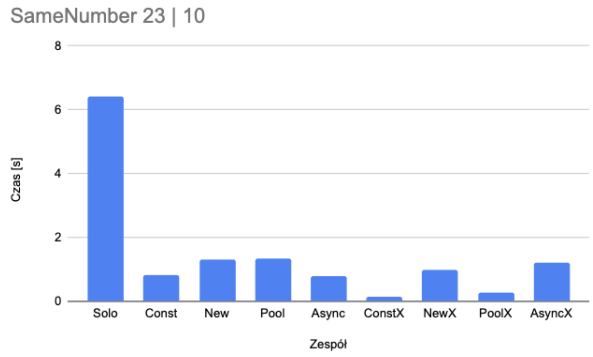


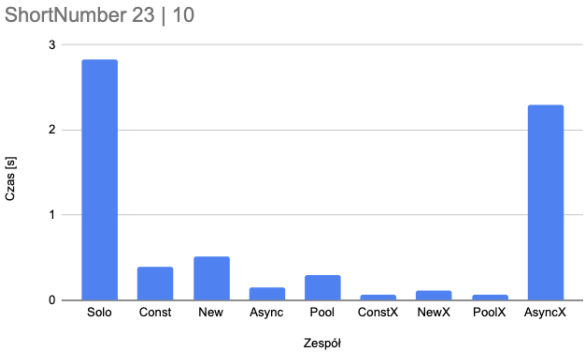
2) Team.size = 10

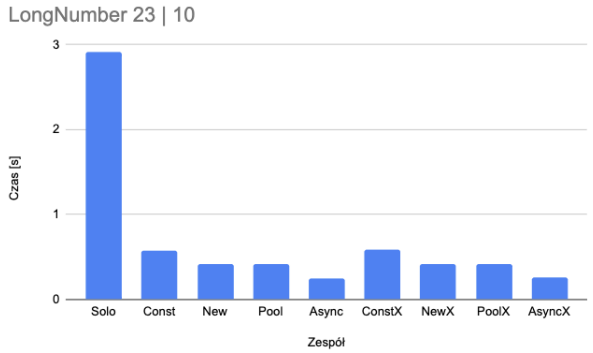
Z tego samego powodu co w 2), TeamNew zachowuje się już lepiej.

Nie mam pojęcia czemu TeamAsyncX zachowuje się tak, jak się zachowuje w konkursie ShortNumber23

Im większy jest rozmiar zespołu, tym naturalnie szybciej wykonuje się program. Zespoły X spędzają bardzo mało czasu we współdzielonej strukturze, więc ich ilość nie pogarsza rezultatów.







Wyniki pochodzą z przeprowadzonych testów na maszynie students.

Dla porównania testy puszczono na własnym laptopie z 6 rdzeniami.(TeamAsyncX zachowuje się normalnie!)