Raport do 3 zadania zaliczeniowego na przedmiocie Programowanie Współbieżne.

Do rozwiązania dołączyłem dodatkowy plik CalcCollatzShared.hpp zawierający zmodyfikowane funkcje CalcCollatz dla drużynX. W pliku Shared results umieściłem lkko zmodyfikowany kod read-writer locka jaki napisałem do drugiego zadania zaliczeniowego. Jako mechanizmu dostępu odpowiedniej ilości wątków w drużynie TeamNewThreads użyłem własnej klasy Semaphore z wykorzystaniem mutexów oraz zmiennych warunkowych. Dla TeamNewProcesses skorzystałem z funkcji wait(0). W SharedResults przechowuję mapę wartości InfInt oraz PromiseFuture będący klasą mieszczącą w sobie zarówno promise jak i future ze standardu C++. Wątki lub procesy dla każdej nowo obliczonej wartości sprawdzają czy jest/ma zostać obliczona przez inny wątek. Jeśli tak to pobierają future z wynikiem obliczeń procesu, który zadeklarował się że to zrobi, natomiast w przeciwnym wypadku, deklarują wykonanie obliczeń dodając do mapy nowy obiekt PromiseFuture z kluczem równym liczbie dla której wynik chcą znaleźć, i zapamiętują to, aby po wykonaniu obliczeń uzupełnić dane obietnice. Zdecydowałem się nie korzystać z pliku new\_process, gdyż moje rozwiązanie drużyn z procesami bez jego użycia wydaje się lepsze. Próbowałem osiągnąć chociaz prymitywną obsługę drużyn TeamNewProcessesX i TeamConstProcessesX, polegającą ona na zapisywaniu wyników jedynie od 1 do 10^6 - 1 w zmapowanej pamięci o z góry ustalonym rozmiarze, jednak ze względu na trudności techniczne głównie w zakresie synchronizacji procesów nie udało mi się tego osiągnąć przed wyznaczonym czasem. Po odkomentowaniu funkcji CallCollatzSharedForProcesses oraz zmianie stałej FAILED\_TRY na true program powinien być w stanie w jakim zakończyłem prace nad drużyną NewProcessesX. Zdaję sobie sprawę z ostrzeżenia jakie pokazuje się przy wywołaniu make, jednak metoda, którą zastosowałem w linijce przed którą ostrzega kompilator została pokazana na laboratoriach więc zakładam, że jest metodą poprawną.

Oto wykresy przedstawiające wyniki dla drużyn z tylko 1 dostępnym wątkiem/procesem dla reprezentacyjnych testów 5 dla każdej drużyny. Czas podany jest w milisekundach. Dodatkowo umieściłem także wykres ze średnią czasów dla testów z każdej ilości wątków dla każdej drużyny. Dla porównania wyników wyników kiedy współbieżność odgrywa faktyczną rolę dodałem również dla porównania w przypadku 10 wątków/procesów gdzie czasy działania znacząco przyspieszyły. Dla 1 wątku/procesu wszystkie drużyny mają gorszy czas od TeamuSolo, niektóre co prawda zbliżony do niego ale wciąż gorszy. Jest to spowodowane dodatkowym czasem poświęconym na uruchomienie mechanizmów współbieżności z których programy te nie korzystają w celu przyspieszenia. Najbardziej zwalnia tutaj NewProcesses oraz NewThreads gdyż tworzą one nowy proces/wątek bardzo wiele razy(tyle ile jest danych) i, szczególnie w przypadku nowego procesu, jest to kosztowne. W przypadku 10 wątków sytuacja ta poprawia się, chociaż wciąż TeamNewProcesses z powodu kosztu tworzenia procesu odstaje od innych. Zmniejsza się natomiast różnica między drużynami zwykłymi a drużynami X która dla 10 wątków mniej przyspiesza programy ze względu na fakt, że niektóre wątki muszą czekać na zakończenie obliczeń. Ogólnie najlepiej wypada drużyna ConstThreadsX ze względu na zastosowanie wszystkich mechanizmów przyspieszających przy jednoczesnej taniej ich obsłudze (nie tworzy wielu wątków, nie korzysta zbyt wiele z semaforów). Niewiele odstaje także NewThreadsX oraz PoolX. Prawie wszystkie drużyny średnio poradziły sobie lepiej od TeamuSolo poza NewProcesses ze względu na duży koszt tworzenia procesu oraz async, prawdopodobnie ze względu na specyfikę działania mechanizmów asynchronicznych. Co ciekawe dla dużych liczb (test long numbers) drużyny korzystające z shared results wypadały gorzej od tych niekorzystających. Jest to spowodowane tym że wątki poświęcały czas na zapis danych, które miały małą szansę zostać wykorzystane przez inny proces co mogłoby spowodować jego przyspieszenie. Zdecydowałem się nie umieszczać wykresów dla testów o id 2 oraz 23 ze względu na to, że zajęłyby one dużo miejsca, a dodanie tylko niewielu nie pozwoli wywnioskować żadnych interesujących danych.

(Mimo jedynek w nawiasach trójkątnych jest to średnia dla wszystkich opcji wartości size; jedynki są pozostałością przy obsłudze danych i tworzeniu wykresu.)

Na kolejnych stronie dodatkowo zamieszczam uśrednione wyniki pochodzące z komputera stacjonarnego (procesor Intel® Core™ i5-7400 CPU @ 3.00Ghz, 4 rdzenie) Niestety nie byłem w stanie przedstawić w raporcie kompletnych danych ze względu na dużą objętośc wykresów.

Jak widać na komputerze stacjonarnym czasy były o wiele mniejsze niż na students. Może być to spowodowane dużym obciążeniem students ze względu na końcowy termin oddawania tegoż projektu. Wszystkie spostrzeżenia, które poczyniłem wcześniej są jednak prawdziwe zarówno na students jak i na komputerze stacjonarnym. Zdecydowałem się przekroczyć limit 2 stron umieszczając dodatkowe wykresy, gdyż czułem, że w raporcie znajduje się zbyt mało danych.

Adam Wojciechowski

aw432823