

Cel: Nabycie umiejętności tworzenia i implementacji programów równoległych w środowisku *Pthreads*

Zajęcia:

1. Utworzenie katalogu roboczego (np. *lab_5*), skopiowanie ze strony przedmiotu paczki *pthreadsum.tgz*, rozpakowanie w podkatalogu (np. *zad_1*), uruchomienie (w miejsce dostarczonej biblioteki *libpomiar_czasu.a* można umieścić własną bibliotekę utworzoną w ramach laboratorium „pomiar_czasu”)
 - a) **każdorazowo (dotyczy to wszystkich programów w ramach wszystkich laboratoriów) należy sprawdzać poprawność działania – wersja równoległa ma dawać identyczne wyniki jak wersja sekwencyjna (jeśli ze względu na sposób dekompozycji wyniki nieznacznie się różnią należy wyjaśnić skąd biorą się różnice)**
 - b) **sens obliczeń równoległych polega na skróceniu czasu realizacji zadania – każdorazowo (dotyczy to wszystkich programów w ramach wszystkich laboratoriów) należy sprawdzić czy udało się uzyskać w wersji równoległej skrócenie czasu działania w stosunku do wersji sekwencyjnej, jeśli użycie rosnącej liczby wątków/procesów nie powoduje skracania czasu działania programu należy wyjaśnić dlaczego tak się dzieje)**
 - c) w przypadku prostego programu obliczania sumy wyrazów tablicy eksperymentem testującym sensowność użycia wersji równoległej może być porównanie czasu wykonania sekwencyjnego i równoległego (np. z użyciem 2, 4 wątków) dla trzech rozmiarów tablic: 1000, 1000000 i 1000000000 – kiedy zastosowanie wersji równoległej ma najsilniejsze uzasadnienie? (wnioski powinny znaleźć się w sprawozdaniu) **(ocena)**
- -> *Jak wygląda prosty wzorzec zrównoleglenia pętli? (podaj odpowiedni fragment procedury wątku i omów działanie wątku, zwracając uwagę na rodzaj dekompozycji pętli) Jakie uzyskuje się czasy obliczeń zwiększając liczbę wątków? Czy zależy to od wielkości zadania?*
2. Pobranie ze strony przedmiotu plików *pthreadcalka.c* i *Makefile*. *pthreadcalka.c* zawiera szkielet do testowania wariantów zrównoleglenia obliczania całki w zadanym przedziale metodą trapezów (dla ułatwienia sprawdzenia poprawności działania domyślnie obliczana jest całka z $\sin(x)$ w przedziale $[0, \pi]$ – wynik powinien być przybliżeniem 2.0). Plik zawiera wersję sekwencyjną obliczania całki, podobną do algorytmu:

```
calka=0.0; dx = (b-a)/N;
for(i=0; i<N; i++){
    x1=a+i*dx;
    calka+=0.5*(funkcja(x1)+funkcja(x1+dx))*dx;
}
```

uwaga: w kodzie każda wartość funkcji (z wyjątkiem skrajnych) jest liczona dwa razy, a mnożenie przez stałą podstawę odbywa się wielokrotnie w sumowaniu – kod nie jest optymalny
3. Kompilacja (*make*) i uruchomienie programu. Przetestowanie działania programu dla różnych zadanych dokładności całkowania (różne wartości dx i wynikające z nich wartości N – uwaga: przy obliczaniu N dokonuje się zaokrąglenia (w efekcie $N \cdot dx \neq (b-a)$), żeby otrzymać dx dokładnie odpowiadające N , należy je ponownie obliczyć – w kodzie obliczana jest nowa wartość dx_{adjust} , taka że $N \cdot dx_{adjust} = (b-a)$ – co odpowiada podziałowi przedziału $[a, b]$ na N podprzedziałów). Sprawdzenie zbieżności wyniku do 2.0 w przypadku malejącej wysokości pojedynczego trapezu (np. $dx = 0.1, 0.01, 0.001, 0.0001$ itd.) i rosnącej w związku z tym liczby trapezów (dla $dx=0.000001$ i $N=3141593$ wynik powinien mieć błąd mniejszy niż 10^{-12} – dwanaście cyfr znaczących) **(ocena)**
4. Uzupełnienie programu *pthreadcalka.c* o wariant obliczania całki w sposób wielowątkowy (z wykorzystaniem środowiska *Pthreads*) wykorzystując **wzorzec zrównoleglenia pętli** (jest to wzorzec do wykonania automatycznego – nie jest potrzebna wiedza, że wykonuje się akurat całkowanie (zawartość wnętrza pętli z kodu sekwencyjnego można dosłownie przenieść do pętli w kodzie równoległym – dla zmiennych lokalnych))
 - a) program główny wywołuje funkcję *calka_zrownoleglenie_petli*, która ma zwrócić wartość obliczonej całki. Funkcja *calka_zrownoleglenie_petli* ma utworzyć zadaną przez użytkownika liczbę wątków, zlecić im wykonanie funkcji *calka_fragment_petli_w*, przesyłając jako argument

identyfikator wątku i oczekiwać na zakończenie pracy wątków – można skorzystać z fragmentów pliku *pthreadsum.c* (odpowiednio je modyfikując)

- b) funkcja ma zrealizować wzorzec zrównoleglenia pętli, np. korzystając ze szkieletu zawartego w pliku *pthreadcal.c*, ostateczny wynik jest uzyskiwany przez redukcję wielowątkową, podobnie jak w *pthreadsum.c* (uwaga na poprawne stosowanie zmiennych prywatnych i wspólnych – wariant w którym wątki otrzymują jako argument wykonywanej funkcji tylko swój identyfikator wymusza użycie wielu zmiennych wspólnych, np. globalnych, co można uznać za niekorzystne z punktu widzenia inżynierii oprogramowania)
- c) liczba iteracji pętli (N) w wariancie sekwencyjnym i równoległym (ze zrównolegleniem pętli) musi być identyczna, wysokość trapezów (dx) także jest identyczna – wynik wersji równoległej musi być identyczny z wersją sekwencyjną (z dokładnością bliską słowu maszynowemu – w praktyce ok. 10^{-12})
- d) dla celów dydaktycznych, w funkcji wątków, należy zastosować zmienne określające indeks początkowy iteracji, indeks końcowy iteracji oraz skok zmiennej sterującej pomiędzy iteracjami, obliczone na podstawie identyfikatora wątku. Należy zastosować wariant cykliczny dekompozycji pętli (dystrybucji iteracji), zapewniając równomierny podział iteracji między wątki (także w przypadku gdy liczba trapezów jest niepodzielna przez liczbę wątków) (**ocena**)
- e) po otrzymaniu poprawnego wyniku dla wariantu cyklicznego dekompozycji, należy zastosować dekompozycję blokową (najlepiej zakomentować podstawienie do zmiennych określających indeks początkowy iteracji, indeks końcowy iteracji oraz skok zmiennej sterującej pomiędzy iteracjami dla dekompozycji cyklicznej i utworzyć kod z podstawieniem do kolejnych zmiennych wartości odpowiednich dla dekompozycji blokowej. (**ocena**)
 - -> *Dlaczego wynik wersji równoległej według wzorca zrównoleglenia pętli jest (nieomal) identyczny z wynikiem wersji sekwencyjnej? Jak można zauważyć wady wersji ze zrównolegleniem pętli?*

3.0

5. Uzupełnienie programu o wariant obliczania całki w sposób wielowątkowy (z wykorzystaniem środowiska *Pthreads*) wykorzystując **wzorzec dekompozycji w dziedzinie problemu**:

- a) program główny wywołuje funkcję *calka_dekompozycja_obszaru*, która ma zwrócić wartość obliczonej całki. Funkcja *calka_dekompozycja_obszaru* ma utworzyć zadaną przez użytkownika liczbę wątków, zlecić im wykonanie funkcji *calka_podobszar_w* i czekać na zakończenie pracy wątków
- b) idea działania funkcji *calka_dekompozycja_obszaru* polega na podziale obszaru obliczeniowego na podobszary i potraktowaniu całkowania w każdym podobszarze jako odrębnego zadania do wykonania przez wątek (zadania są wykonywane niezależnie – nie ma komunikacji, synchronizacji, mutexów, itp.)
- c) każdy wątek otrzymuje zadanie wycalkowania funkcji w podobszarze: nie musi znać swojego identyfikatora (choć może), musi tylko znać dokładne parametry całkowania w podobszarze: współrzędnego lewego i prawego krańca podobszaru oraz zadaną dokładność w postaci wysokości pojedynczego trapezu dx (na podstawie dx oblicza własne N_{local} i poprawia dx – tak jak program *main* z dx_{adjust} , uwaga: czy wartości dx_{adjust_local} są takie same we wszystkich wątkach?)
- d) w praktyce każdy wątek otrzymuje jako argument wykonywanej funkcji wskaźnik do odpowiedniej struktury z zadanymi parametrami wejściowymi, przeprowadza obliczenia i przekazuje wynik – np. może zapisać go w specjalnie zaprojektowanym polu struktury stanowiącej argument
- e) jako zadanie do samodzielnego wykonania pozostaje poprawne zaimplementowanie przesłania wyniku całkowania z *calka_podobszar_w* do wiadomości *calka_dekompozycja_obszaru* (z synchronizacją między wątkami lub bez), tak żeby funkcja *calka_dekompozycja_obszaru* mogła zwrócić ostateczny wynik (można wykorzystać sugerowaną postać struktury zawierającej definicję zadania dla konkretnego wątku) (**ocena**)
 - -> *Jak wygląda kod w wersji dekompozycji w dziedzinie problemu (dekompozycji obszaru całkowania)? (podaj odpowiedni fragment funkcji wątku i omów go). Dlaczego wynik w wersji dekompozycji obszaru może różnić się od wyniku sekwencyjnego? (podpowiedź: jaki warunek spełniają dx_{adjust} , a jaki dx_{adjust_local} ? - co z tego wynika?)*

4.0

Dalsze kroki dla podniesienia oceny:

1. Wykorzystanie do zwracania wyniku z funkcji realizowanych przez wątki argumentu funkcji *pthread_exit*, który następnie odczytywany jest przez *pthread_join* (uwaga na rzutowanie typów).
2. Optymalizacja obliczania całki w pętli, np. tak żeby liczyć każdą wartość funkcji tylko raz, a dodatkowo zmniejszyć liczbę operacji wyciągając mnożenie przed sumowanie (modyfikacja np. dla wersji równoległej z dekompozycją obszaru)
3. Dowolne zadania polecane przez prowadzących

----- 5.0 -----

Warunki zaliczenia:

- Obecność na zajęciach i wykonanie kroków 1-4
- Oddanie sprawozdania o treści i formie zgodnej z regulaminem ćwiczeń laboratoryjnych – z opisem zadania, kodem źródłowym programów i analizą wyników dla wszystkich wariantów (wydruki wyników wklejone jako obrazy z identyfikacją osoby przeprowadzającej obliczenia – zgodnie z regulaminem laboratoriów) oraz odpowiednimi wnioskami.
- Symbol -> oznacza pytania, na które odpowiedzi ma dać laboratorium (odpowiedzi powinny znaleźć się w sprawozdaniu – najlepiej w punktach odpowiadających pytaniom)