Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Laboratorium 12.

Cel: Doskonalenie podstaw programowania z przesyłaniem komunikatów MPI.

Kroki:

1. Utworzenie katalogu roboczego (np. *lab12*) i podkatalogu (np. *MPI_pi*).
2. Opracowanie programu obliczającego liczbę π z szeregu Leibniza:

   $$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4}$$,

   Proces o randze 0 powinien pobrać informację o liczbie sumowanych składników (podaną jako parametr przy uruchomieniu programu, z klawiatury itp.).
   Liczba obliczanych składników szeregu powinna zostać równo rozdzielona między procesy liczące sumy częściowe w celu zrównoważenia obciążenia (należy rozwiązać problem w przypadku niepodzielności liczby składników przez liczbę procesów liczących).
   1. Na stronie przedmiotu znajduje się plik z wersją sekwencyjną obliczania liczby π – *oblicz_PI.c*
   2. Napisanie kodu równoległego sprowadza się do zrównoleglenia pętli obliczającej π
   3. Napisanie własnego kodu może polegać na następujących krokach:
      1. napisanie standardowego szkieletu z funkcjami pobierającymi rangę procesu i rozmiar komunikatora oraz inicjującymi i finalizującymi MPI (można wykorzystać wzór z przykładu obliczania całki na slajdach do wykładu 11 (slajd 9) lub program *MPI_simple.c* z poprzedniego laboratorium)
      2. umieszczenie wymiany komunikatów odpowiedniej do rozwiązywanego problemu obliczania π (wzór z wykładu daje bardzo zbliżoną podpowiedź – należy zwrócić uwagę na istotne różnice między użyciem komunikacji grupowej, gdzie w identyczny sposób funkcja jest wywoływana przez wszystkie procesy, a wymianą punkt-punkt, gdzie występuje asymetria *send* i *receive*, jak to ma miejsce np. w programie z *MPI_simple.c*)
      3. zrównoleglenie pętli dokonać w sposób podobny jak w *pthreads* (jednak tym razem w obliczeniach, zgodnie z filozofią MPI mają uczestniczyć wszystkie procesy, bez podziału na zarządcę i wykonawców):
         1. na podstawie swojej rangi i całkowitej liczby procesów (rozmiaru komunikatora) każdy proces indywidualnie ustala, które iteracje ma wykonać (*my_start, my_end, my_stride; czyli moj_poczatek, moj_koniec, moj_skok* – można jak zwykle dokonać dekompozycji cyklicznej lub blokowej – w przypadku obliczania π dekompozycja blokowa może dawać wyniki dokładniejsze ze względu na częściowe unikanie błędów zaokrągleń)
         2. treść pojedynczej iteracji jest identyczna jak w wersji sekwencyjnej (z pliku *oblicz_PI.c*)
      4. Proces o randze 0 ma wczytywać dane początkowe (liczbe iteracji) i uzyskać ostateczny wynik
      5. Do kompilacji można użyć zmodyfikowanego pliku *Makefile* z *MPI_simple.tgz*
3. Testowanie opracowanego programu (sprawdzenie poprawności otrzymanego wyniku – wydruk wyniku, z porównaniem z wartością biblioteczną M_PI, powinien pojawić się w procesie o randze 0). **(ocena)**
4. Utworzenie podkatalogu (np. *MPI_mat_vec*)
5. Pobranie paczki *MPI_mat_vec_row.tgz,* rozpakowanie, uruchomienie kodu
   - weryfikacja poprawności obliczeń mnożenia macierz-wektor *Ax* (brak wydruku o ewentualnych błędach) oraz zysku czasowego w stosunku do wersji sekwencyjnej
6. Analiza kodu, wyróżnienie fragmentów realizujących:

- inicjowanie danych MPI
- inicjowanie danych wejściowych mnożenia macierz-wektor w procesie o randze 0 oraz wykonanie sekwencyjne, także w procesie o randze 0, algorytmu mnożenia macierz-wektor, którego wynik (wektor *y*) służy potem do weryfikacji poprawności algorytmu równoległego MPI
- wykonanie równoległe algorytmu mnożenia macierz-wektor z wymianą komunikatów MPI (linie 69-194)
  - rozesłanie parametrów zadania przez proces o randze 0: rozgłoszenie rozmiaru macierzy oraz rozproszenie samej macierzy *A* i wektora *x*
  - wykonanie wzajemnego przekazania fragmentów wektora *x* przez wszystkie procesy
    - w przypadku badanego algorytmu jest to zbędne, jednak jest konieczne jeśli fragmenty wektora *x* nie są wstępnie rozpraszane, ale są generowane indywidualnie przez wszystkie procesy
  - obliczenia lokalne dla każdego z procesów (klasyczna realizacja modelu SPMD)
  - zebranie lokalnych wyników (z lokalnych wektorów *z*) do globalnego wektora *z* w procesie o randze 0
- sprawdzenie poprawności obliczeń (porównanie wektorów *y* i *z*)
- dalszy ciąg (od linii ok. 207) jest przygotowaniem do realizacja zadania dekompozycji kolumnowej (zadanie rozszerzające na 5.0 - poniżej)

7. Modyfikacja kodu polegająca na zamianie wymiany komunikatów za pomocą *MPI_Send* i *MPI_Recv* na procedury komunikacji grupowej
   1. komunikacja dotyczy rozsyłania danych początkowych z procesu o randze 0 do innych procesów i odbierania wyniku przez proces o randze 0 od innych procesów
   2. należy dobrać właściwe procedury komunikacji grupowej i odpowiednio zaimplementować ich wywołanie
      - *w czym określanie wysyłanych i odbieranych danych (czyli ustalanie, które elementy danych wysyła i otrzymuje konkretny proces) w sekwencji komunikatów punkt-punkt różni się od określania w komunikacji grupowej? jakie założenie robi MPI w przypadku komunikacji grupowej?*

        (podpowiedź: w dostarczonym kodzie określanie położenia danych, dla każdego z procesów – jako zależne od jego rangi – jest zgodne z konwencją MPI, ale mogłoby być inne – sekwencja komunikatów punkt-punkt daje większą swobodę określania położenia danych dla każdego z procesów niż wykorzystanie komunikacji grupowej)
   3. realizację zadania można rozpocząć od zamiany tylko dla procedury zbierania wyniku
      - w dostarczonym kodzie zbieranie danych (składowych wynikowego wektora *z*) jest dokonywane za pomocą sekwencji komunikatów *point-to-point* w linijkach ok. 183-194
      - przystępując do modyfikacji kodu można zakomentować wymianę komunikatów, zaprojektować wywołanie procedury komunikacji grupowej i wykorzystać znajdujące się w kolejnych liniach 196-204 sprawdzenie wyniku, które w wydrukach podaje poprawną wartość dla każdej składowej
        - w celu uniknięcia problemów ze zbyt dużą liczbą wydruków zaciemniających sytuację, pracę nad wersją z komunikacja grupową można przeprowadzić dla małej macierzy o wymiarze np. 16 (dla 2 lub 4 procesów MPI)
8. Sprawdzenie poprawności działania kodu po modyfikacji (proces o randze 0 wykonuje obliczenia sekwencyjne i porównuje wyniki) **(ocena)**

---------- 3.0 - zebranie wynikowego wektora *z* ----------

---------- 3.5 - rozproszenie danych wejściowych i zebranie wynikowego wektora *z* ----------

Kroki dodatkowe:

1. W zadaniu 6 rozważenie przypadku kiedy procesem dokonującym rozproszenia i zbierania danych nie jest proces o randze 0 (wykorzystanie tego procesu ma tę zaletę, że jest zawsze obecny przy wykonaniu programu MPI), ale inny proces, np. o randze 1 (ten proces jest zawsze obecny przy wykonaniu z liczbą procesów MPI większą od 1)
    1. należy uwzględnić to przy alokacji i wypełnianiu tablic z danymi oraz przy właściwych procedurach komunikacji
    2. należy uwzględnić dwie wersje: pierwszą z jawnym wskazywaniem lokalizacji danych dla każdego procesu do odczytu i zapisu w procedurach komunikacji grupowej, i drugą z wykorzystaniem opcji określenia jednego z buforów za pomocą symbolu MPI_IN_PLACE, wskazującego, że dla procesu *root* dane pozostają w tym samym miejscu – nie są nigdzie przepisywane (z postaci procedury i określenia drugiego z buforów implementacja MPI sama wnioskuje o adresie bufora zastąpionego przez MPI_IN_PLACE)

---------- 4.0 ----------

2. Modyfikacja kodu mnożenia macierz-wektor dla kolumnowej dekompozycji macierzy (macierz powinna nadal być przechowywana wierszami)
    - należy zastosować wskazówki zawarte w kodzie (od linii ok. 208)
    - rozważenie uzyskania wyniku poprzez MPI_Allreduce
        - każdy proces oblicza wynik w swojej kopii wektora wynikowego, redukcja dotyczy każdego elementu wektora
    - rozważenie uzyskania wyniku poprzez MPI_Alltoall
        - każdy proces oblicza wynik w swojej kopii wektora, po wymianie all-to-all wektor zawiera fragmenty pochodzące od różnych procesów, ułożone w kolejności rang – należy lokalnie obliczyć wartości kolejnych elementów wektora wynikowego (każdy proces dla swojego fragmentu wektora) jako sumę udziałów od poszczególnych procesów
3. Modyfikacja kodu mnożenia macierz-wektor dla kolumnowej dekompozycji macierzy i macierzy przechowywanej kolumnami – zmiana przechowywania macierzy ma wpływ tylko na sposób rozpraszania macierzy i przeprowadzania lokalnych obliczeń, sposób postępowania z wektorem wynikowym pozostaje taki sam jak przy przechowywaniu wierszami
    1. rozważenie uzyskania wyniku poprzez MPI_Allreduce
    2. rozważenie uzyskania wyniku poprzez MPI_Alltoall

Warunki zaliczenia:

1.Obecność na zajęciach i wykonanie kroków 1-8
2.Oddanie sprawozdania z opisem zadania, kodem źródłowym programów, wynikami i wnioskami – zgodnie z regulaminem laboratoriów.