

do zdobycia: [20pkt]
deadline: 18/23 III 2015

Zestaw 2

Tym razem zajmiemy się już nieco bardziej złożonym problemem. Naszym zadaniem jest napisanie programu równoległego służącego do (niezbyt złożonego) przetwarzania wektorów.

Nasz problem można sformułować w sposób następujący:

1. dane jest N wektorów z przestrzeni \mathbb{R}^3 , wektory te oznaczmy jako $\mathbf{r}_i = [x_i, y_i, z_i]$, oczywiście $i = 1, 2, \dots, N$,
2. należy obliczyć średnią długość l oraz średni wektor $\langle \mathbf{r} \rangle$, dane są one wzorami

$$l = \frac{1}{N} \sum_{i=1}^N |\mathbf{r}_i| = \frac{1}{N} \sum_{i=1}^N \sqrt{x_i^2 + y_i^2 + z_i^2} \quad (1)$$

oraz

$$\langle \mathbf{r} \rangle = [\langle x \rangle, \langle y \rangle, \langle z \rangle], \quad (2)$$

gdzie

$$\langle q \rangle = \frac{1}{N} \sum_{i=1}^N q_i \quad (3)$$

oraz $q = \{x, y, z\}$.

Poniżej (w punktach) przedstawiona została specyfikacja programu, który należy napisać. W ramach kolejnych dwóch ćwiczeń będziemy rozwijać (ulepszać) ten program. Warto więc od samego początku zadbać, by jego kod był przemyślany i przejrzysty.

1. Na wstępie program powinien wczytać dane (współrzędne wektorów) z pliku (tekstowego). Załóżmy, że plik z danymi posiada następujący format (symbolicznie):

```
<float x_1> <float y_1> <float z_1>
<float x_2> <float y_2> <float z_2>
...      ...      ...
<float x_N> <float y_N> <float z_N>
```

Jak widać, w i -tej linii pliku określone są składowe wektora \mathbf{r}_i . Na wstępie przyjmiemy dość proste podejście, w którym **każdy** z procesów będzie wczytywał **całość** danych z pliku. Należy przyjąć, że przy uruchomieniu programu nie znana jest ilość danych w pliku (tj. liczba wektorów N). Program powinien ustalić wartość N w trakcie czytania pliku z danymi. Dla uproszczenia można jednak przyjąć, że $1 \leq N \leq 10^6$.

UWAGA: do czytania danych z pliku **nie stosujemy** (póki co...) mechanizmów dostarczanych przez MPI.

2. W etapie następnym powinna nastąpić dekompozycja problemu. Dane wejściowe należy podzielić na n (w miarę możliwości) równych bloków. W zależności od wartości n (liczba procesów) oraz wartości N (liczba wektorów) program powinien ustalić wartości i_k^0 oraz i_k^1 . Symbole i_k^0 oraz i_k^1 oznaczają indeksy pierwszego oraz ostatniego wektora, o które „troszczy się” proces k -ty ($k = 0, 1, \dots, n - 1$). Oczywiście, proces k -ty powinien troszczyć się

o wszystkie wektory o indeksach z zakresu od i_k^0 (włącznie) do i_k^1 (włącznie). Koniecznie należy zadbać, by nakład pracy był równomiernie rozdzielany pomiędzy procesy. Koniecznie należy też zadbać, by program działał prawidłowo w przypadkach, w których N nie jest całkowitą wielokrotnością n . Program powinien również działać prawidłowo w sytuacjach, w których $N < n$. Warto solidnie zastanowić się nad algorytmem wyznaczania i_k^0 oraz i_k^1 . Warto również uprościć ten algorytm.

UWAGA: rozważany problem zawsze można zdekomponować tak, że dowolne dwa procesy „dostają” porcje różniące się o nie więcej aniżeli 1 (słownie: jeden) wektor!

WSKAZÓWKA: co wyrażają wartości N / n oraz $N \% n$?

3. W kroku kolejnym każdy z procesów oblicza swój wkład do sum pojawiających się w (1) oraz (3).
4. W etapie następnym redukowane są rezultaty częściowe (obliczone przez poszczególne procesy składowe sum (1) oraz (3)). Zrealizować to należy wykorzystując funkcję `MPI_Reduce`.
5. W ostatnim kroku master oblicza wartość l oraz $\langle \mathbf{r} \rangle$, wypisując rezultaty na ekranie.

Dodatkowo program powinien mierzyć czasy wykonania poszczególnych etapów obliczeń. Do pomiarów czasów należy posłużyć się funkcją `MPI_Wtime`. Program wyszczególniać powinien czasy niezbędne do realizacji poszczególnych etapów, w tym: czas potrzebny na wczytanie danych, czas potrzebny na obliczenie wkładów do średnich, czas potrzebny na redukcję wyników. Po zakończeniu obliczeń program powinien zapisywać informację o czasach wykonania do pliku tekstowego. Jego postać może wyglądać następująco:

timings (proc 0):

readData:	1.24351
processData:	0.00634193
reduceResults:	0.000319004
total:	1.25017

timings (proc 1):

readData:	1.24423
processData:	0.00612283
reduceResults:	0.00022006
total:	1.25057

timings (proc 2):

readData:	0.828303
processData:	0.006042
reduceResults:	0.000135183
total:	0.83448

total timings:

readData:	3.31604
processData:	0.0185068
reduceResults:	0.000674248
total:	3.33522

Plik ten powinien być tworzony przez mastera. Do wymiany informacji o czasach wykonania pomocna może okazać się funkcja `MPI_Gather`.

Przykładowe dane wejściowe znaleźć można (na olimpie) w katalogu `~swinczew/AR`. W katalogu tym znajdują się pliki o nazwach `v01.dat`, `v02.dat`, ..., `v06.dat`. W katalogu tym znajduje się również plik o nazwie `vresults.dat`. Zawiera on prawidłowe wyniki (obliczone dla poszczególnych plików wejściowych wartości l oraz $\langle \mathbf{r} \rangle$). Warto upewnić się, że program zwraca identyczne rezultaty. Warto również upewnić się, że zwracane przez program rezultaty nie zależą od n (liczby procesów).

Kryteria oceniania:

1. rozsądna, prawidłowa i zgrabna metoda dekompozycji danych [4pkt],
2. prawidłowa implementacja części obliczeniowej [4pkt],
3. implementacja funkcjonalności pomiaru czasów wykonania [4pkt],
4. szczegółowa analiza i omówienie czasów wykonania [4pkt],
5. estetyka i struktura kodu [4pkt].