Tomasz Łaz, Ignacy Migas
dr inż. Dydejczyk
19 maja 2018

Parallel sorting by regular sampling

SRIR

1. Opis algorytmu

Algorytm „Równoległego sortowania przez regularne dzielenie” został wymyślony przez Li.

Oryginalnie algorytm składa się z czterech faz:

1. Faza I

   Sortowanie swojego zestawu danych przez każdy proces

2. Faza II

   Jeden z procesów zbiera od innych procesów i siebie wartości równe liczbie procesów, które są wybierane w sposób regularny w każdym z procesów. Kolejno sortuje te elementy i wybiera „liczba procesów - 1” elementów, które przesyłane są do kolejnych procesów. Następnie każdy proces partycjonuje swój zbiór danych na odpowiednią liczbę podzbiorów w oparciu o otrzymane *pivoty* od serwera.

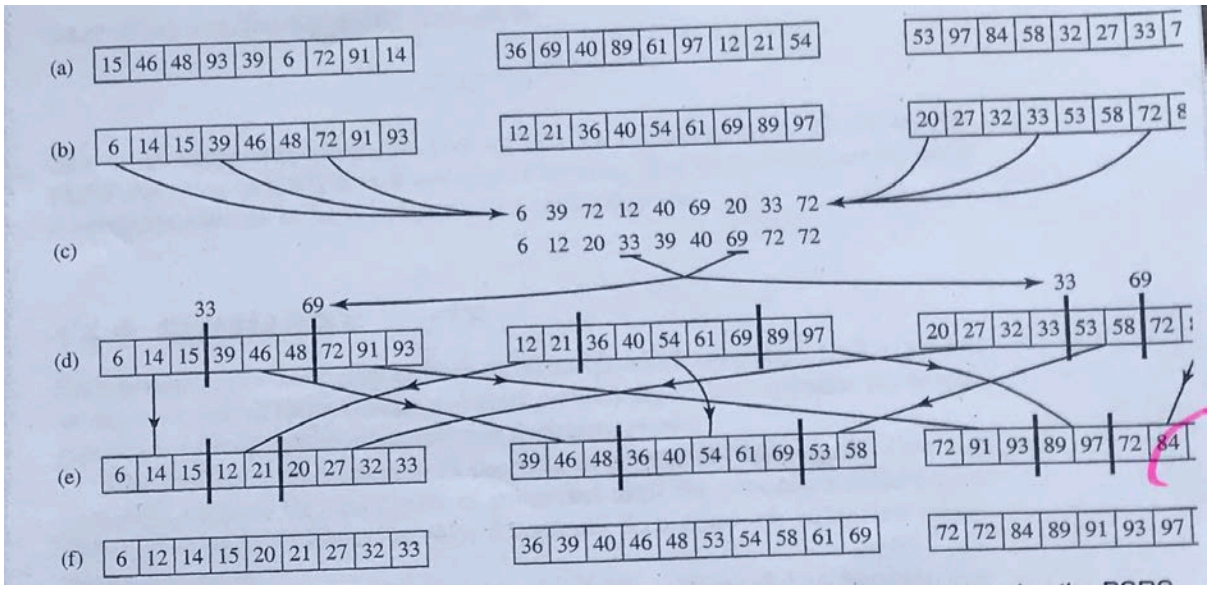
3. Faza III

W tej fazie, każdy z procesów zachowuje „swoją” (przypisaną do danego procesu) porcję danych i przesyła pozostałe dane do reszty procesów.

4. Faza IV

Każdy z procesów łączy zebrane dane z każdego procesu w jedną listę i je sortuje (*multimerge*).

W fazie tej maksymalna liczba elementów, które dany proces musi złączyć wynosi 2 * liczba elementów / liczba procesów



Rys. 1 Diagram przedstawiający przejście algorytmu

Wydajność algorytmu:

p - liczba procesów

n - liczba elementów

Dla obliczeń

O[(n/p) * log(n/p) + p^2 * log (n/p) * log(p)]

Dla n >> p: O[(n/p) * (log(n) + log(p)]

Dla komunikacji:

O(log(p) + n/p)

Dla n >> p: O(n/p)

Funkcja skalowalności:

p^C / p = p^(C-1)

Algorytm ten posiada trzy zalety względem algorytmu *hyperquicksort:*

- Przechowuje listę rozmiarów w bardziej zbalansowany sposób pomiędzy procesami
- Pomija niepotrzebną komunikację pomiędzy kluczami
- Nie wymaga, aby liczba procesów była potęgą liczby 2

Ponadto algorytm umożliwia wprowadzenie prostej komunikacji *all-to-all*, która może być zaimplementowana w taki sposób, aby każdy element był przenoszony tylko raz.

2. Opis struktury projektu

Projekt składa się z 3 plików:

- utilities.h + utilities.c
- PSRC.c

W plikach utilities znajdują się metody wykorzystywane w algorytmie do wypisywania macierzy, porównywania oraz łączenia tablic (multimerge).

Plik PSRC.c to właściwy kod realizujący algorytm, w komentarzach w kodzie mamy wydzielone fazy, które są opisane w punkcie 3.

Ponadto w projekcie umieszczono katalog z przykładowymi danymi: *sample*, gdzie input to dane wejściowe do posortowania, a output wynik oczekiwany

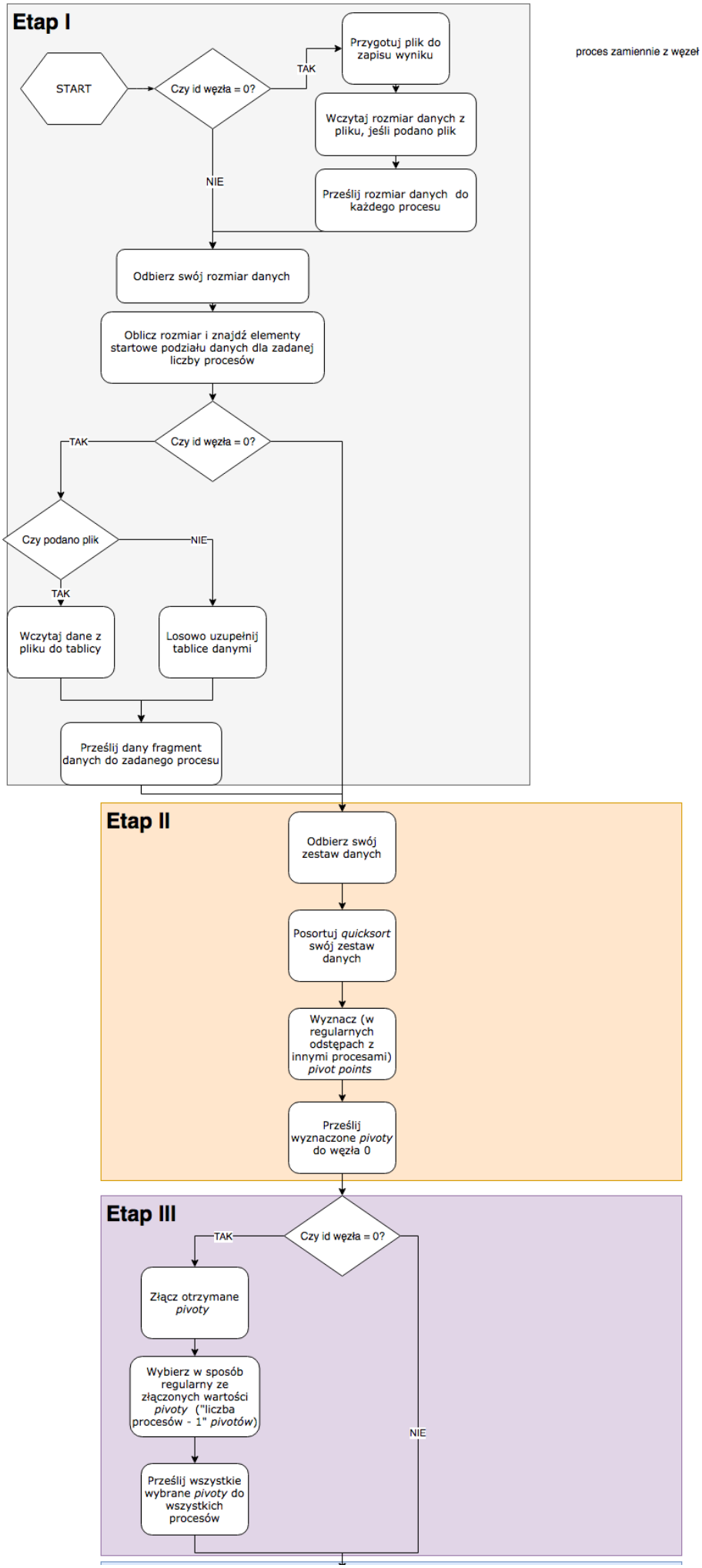
W projekcie znajduje się również katalog doc z dokumentacją.

Ponadto do projektu dołączono makefile, którego metody opisane zostały w punkcie 4

3. Opis działania

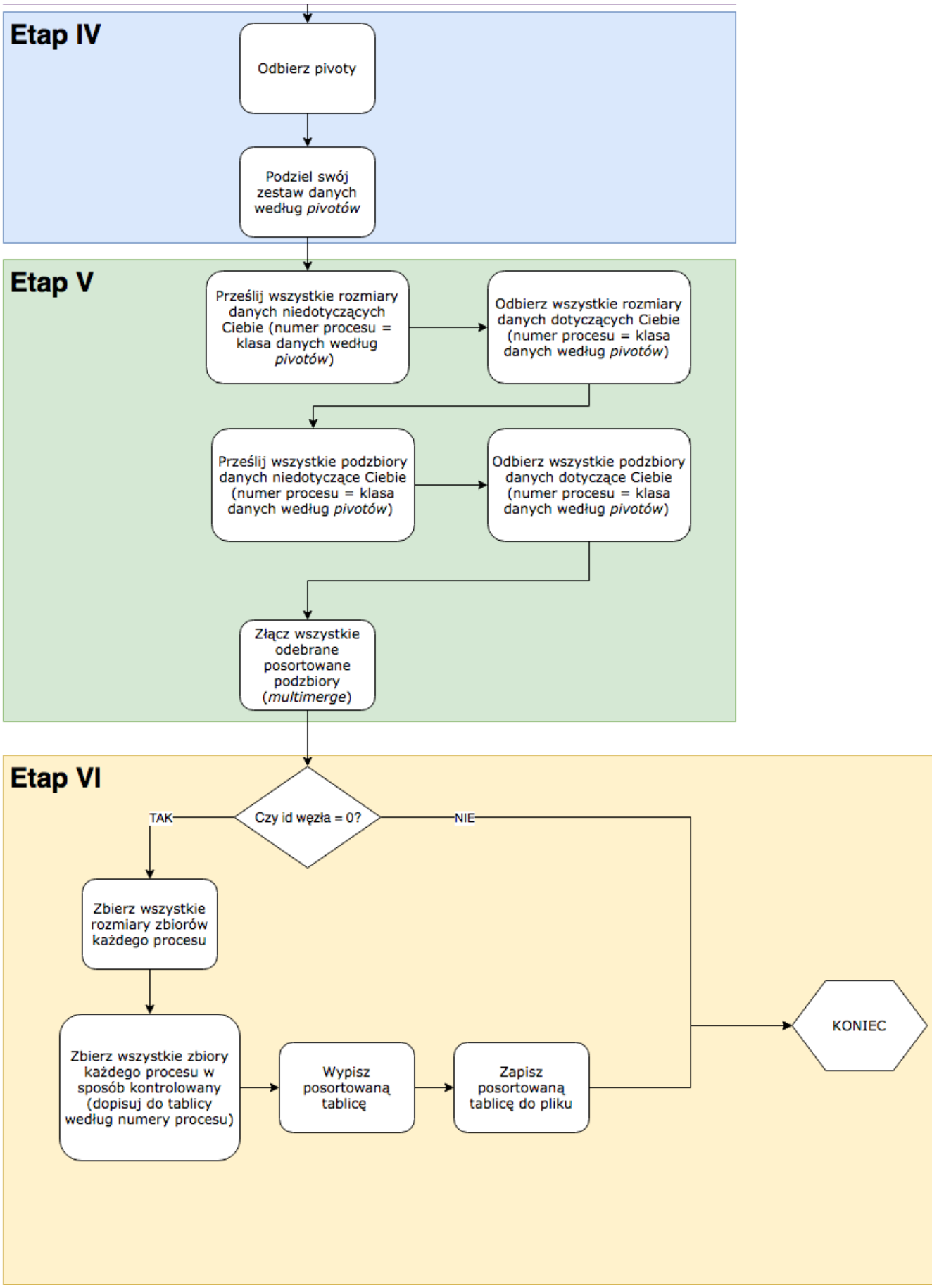
Opis działania przedstawiony został na diagramie Diag. 1, Diag. 2

Diagram został również umieszczony w folderze doc/



proces zamiennie z węzeł

Diag. 1


Etap IV
Odbierz pivoty
Podziel swój zestaw danych według *pivotów*
Etap V
Prześlij wszystkie rozmiary danych niedotyczących Ciebie (numer procesu = klasa danych według *pivotów*)
Odbierz wszystkie rozmiary danych dotyczących Ciebie (numer procesu = klasa danych według *pivotów*)
Prześlij wszystkie podzbiory danych niedotyczące Ciebie (numer procesu = klasa danych według *pivotów*)
Odbierz wszystkie podzbiory danych dotyczące Ciebie (numer procesu = klasa danych według *pivotów*)
Złącz wszystkie odebrane posortowane podzbiory (*multimerge*)
Etap VI
Czy id węzła = 0?
TAK
NIE
Zbierz wszystkie rozmiary zbiorów każdego procesu
Zbierz wszystkie zbiory każdego procesu w sposób kontrolowany (dopisuj do tablicy według numery procesu)
Wypisz posortowaną tablicę
Zapisz posortowaną tablicę do pliku
KONIEC

Aplikacja składa się z 6 etapów, które pokrywają się z etapami algorytmu przedstawionymi w punkcie 1.

1. ETAP1 - Początkowo aplikacja przygotowuje dane do posortowania
2. ETAP 2 - W kolejnej fazie, dane są rozdzielane na procesu i wybierane są wartości, będące *pivotami*
3. ETAP 3 - Następnie pivoty są przesłane do servera, który je łączy i wybiera „liczba procesów - 1” pivotów. Pivoty te są przesyłane do każdego procesu.
4. ETAP 4 - Kolejno pivoty te są wykorzystywane przez każdy proces to podziału swoich danych na partycje
5. ETAP 5 - W następnym kroku każdy proces pozyskuje tylko dane z zadanej przez identyfikator partycji od każdego procesu i je łączy.
6. ETAP 6 - Na końcu dane te są przesyłane do serwera, który je łączy i wypisuje do pliku

Diag.2

4. Opis obsługi programu

Kompilacja programu i jego uruchomienie odbywa się przy wykorzystaniu makefile'a.

Udostępnia on 4 reguły:

- *make all* zbudowanie projektu
- *make clean* przywrócenie folderu do stanu początkowego (usunięcie plików .o oraz .out)
- *make rebuild* wyczyszczenie i zbudowanie projektu
- *make run N=<liczba procesów> FILE==<sciezka do pliku> NODES=<nodes>*

- uruchomienie aplikacji podając odpowiednio parametry: N oraz opcjonalnie FILE i NODES, gdzie
N - liczba procesów (domyślnie 4),
FILE to nazwa pliku do wczytania do procesowania,
NODES to wskazanie na plik z adresami węzłów
np: *make run N=4 FILE==sample/input_50.data NODES=nodes*
- uruchomi program na 4 procesach i na tylu węzłach ile w pliku nodes, wczytując dane z pliku o scieżce sample/input_50.data

Kroki do skompilowania i uruchomienia na jednym węźle:

1. Przejdź do katalogu z projektem i plikiem make: MPI_PSRS
2. Wykonaj *make clean*
3. Wykonaj *make run N=<liczba procesów> FILE=<sciezka do pliku>*

4. Przykład wyjścia z programu

Na rys. 2 oraz rys. 3 przedstawiono wyjście z programu dla danych z pliku.

```
tlaz@tlaz:~/Projects/MPI_PSRS$ mpiexec -n 4 ./PSRS_xmp.out -f sample/input_50.data

Starting
Number of processes 4

Size of whole data to process: 50
[0] Table values to sort:
7 16 3 84 20 17 47 40 3 69 92 96 74 32 53 43 81 73 31 13 91 88 21 98 54 52 36 67 49 78 95 84 75 46 88 41 76 2 83 4 8 77 49 93 40 80 97 18 26 36
The end
[process-0] myDataLength=12, myDataStarts=0
[process-2] myDataLength=12, myDataStarts=24
[process-0] myData 3 3 7 16 17 20 40 47 69 84 92 96
[process-0] pivots 3 16 40 84
[process-1] myDataLength=12, myDataStarts=12
[process-3] myDataLength=14, myDataStarts=36
[process-2] myData 36 41 46 49 52 54 67 75 78 84 88 95
[process-2] pivots 36 49 67 84
[process-1] myData 13 21 31 32 43 53 73 74 81 88 91 98
[process-1] pivots 13 32 73 88
[process-3] myData 2 4 8 18 26 36 40 49 76 77 80 83 93 97
[process-3] pivots 2 18 49 80
[process-0] server pivots 3 16 40 84 13 32 73 88 36 49 67 84 2 18 49 80
[process-0] Merged pivots 2 3 13 16 18 32 36 40 49 49 67 73 80 84 84 88
[process-0] Chosen pivots 18 49 80
[process-3] Chosen pivots on node 18 49 80
[process-3][class-0] class start=0, class length=4
[process-1] Chosen pivots on node 18 49 80
[process-1][class-0] class start=0, class length=1
[process-1][class-1] class start=1, class length=4
[process-1][class-2] class start=5, class length=3
[process-1][class-3] class start=8, class length=4
[process-2] Chosen pivots on node 18 49 80
[process-2][class-0] class start=0, class length=0
[process-2][class-1] class start=0, class length=4
[process-2][class-2] class start=4, class length=5
[process-2][class-3] class start=9, class length=3
[process-0] Chosen pivots on node 18 49 80
[process-0][class-0] class start=0, class length=5
[process-0][class-1] class start=5, class length=3
[process-0][class-2] class start=8, class length=1
[process-0][class-3] class start=9, class length=3
[process-3][class-1] class start=4, class length=4
[process-3][class-2] class start=8, class length=3
[process-3][class-3] class start=11, class length=3
[0] tmp: 0 5 | new: 0 5
[2] tmp: 8 1 | new: 0 1
[1] tmp: 5 3 | new: 0 3
[0] tmp: 0 1 | new: 5 1
[2] tmp: 5 3 | new: 1 3
[2] tmp: 4 5 | new: 4 5
[1] tmp: 1 4 | new: 3 4
[1] tmp: 0 4 | new: 7 4
[2] tmp: 8 3 | new: 9 3
[1] tmp: 4 4 | new: 11 4
[3] tmp: 9 3 | new: 0 3
[0] tmp: 0 0 | new: 6 0
[3] tmp: 8 4 | new: 3 4
[0] tmp: 0 4 | new: 6 4
[3] tmp: 9 3 | new: 7 3
[3] tmp: 11 3 | new: 10 3
[process-2] Data for myid partition 52 53 54 67 69 73 74 75 76 77 78 80
[process-1] Data for myid partition 20 21 26 31 32 36 36 40 40 41 43 46 47 49 49
[process-3] Data for myid partition 81 83 84 84 88 88 91 92 93 95 96 97 98
[process-0] Data for myid partition 2 3 3 4 7 8 13 16 17 18
[process-0] sorted data 2 3 3 4 7 8 13 16 17 18 20 21 26 31 32 36 36 40 40 41 43 46 47 49 49 52 53 54 67 69 73 74 75 76 77 78 80 81 83 84 84 88 88 91 92 93 95 96 97 98

Clock time (seconds) = 0.001031

The end
tlaz@tlaz:~/Projects/MPI_PSRS$
```

Rys. 2 Przykładowe wyjście z programu dla N=4 FILE=sample/input_50.data

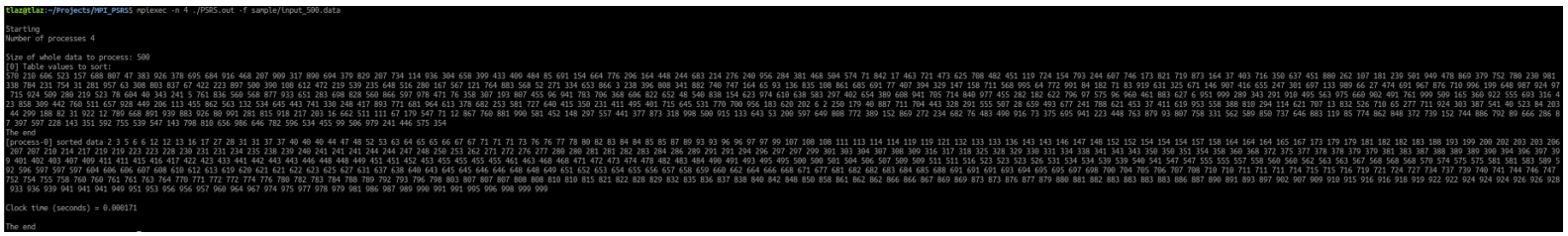
Rys. 3 Przykładowe wyjścia z programu dla N=2 FILE=sample/input_500.data

Wyjście z programu prezentuje:

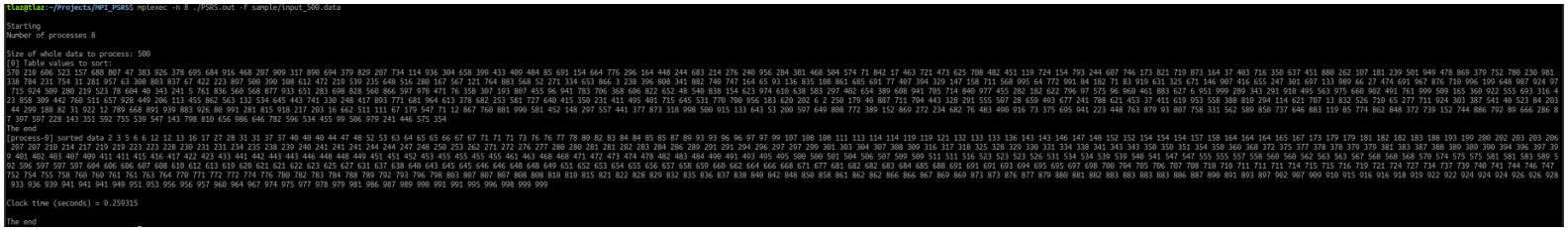
- liczbę procesów,
- rozmiar danych do sortowania
- tablice wejściową z danymi
- rozmiar danych do posortowania na danym procesie wraz z indeksem startowym
- posortowane dane na danym procesie
- pivoty na danym procesie
- pivoty wybrane globalnie
- podział danych na danym procesie ze względu na pivoty wraz z długością danych w danej partycji
- aktualne dane na procesie

- posortowane dane
- czas działania algorytmu

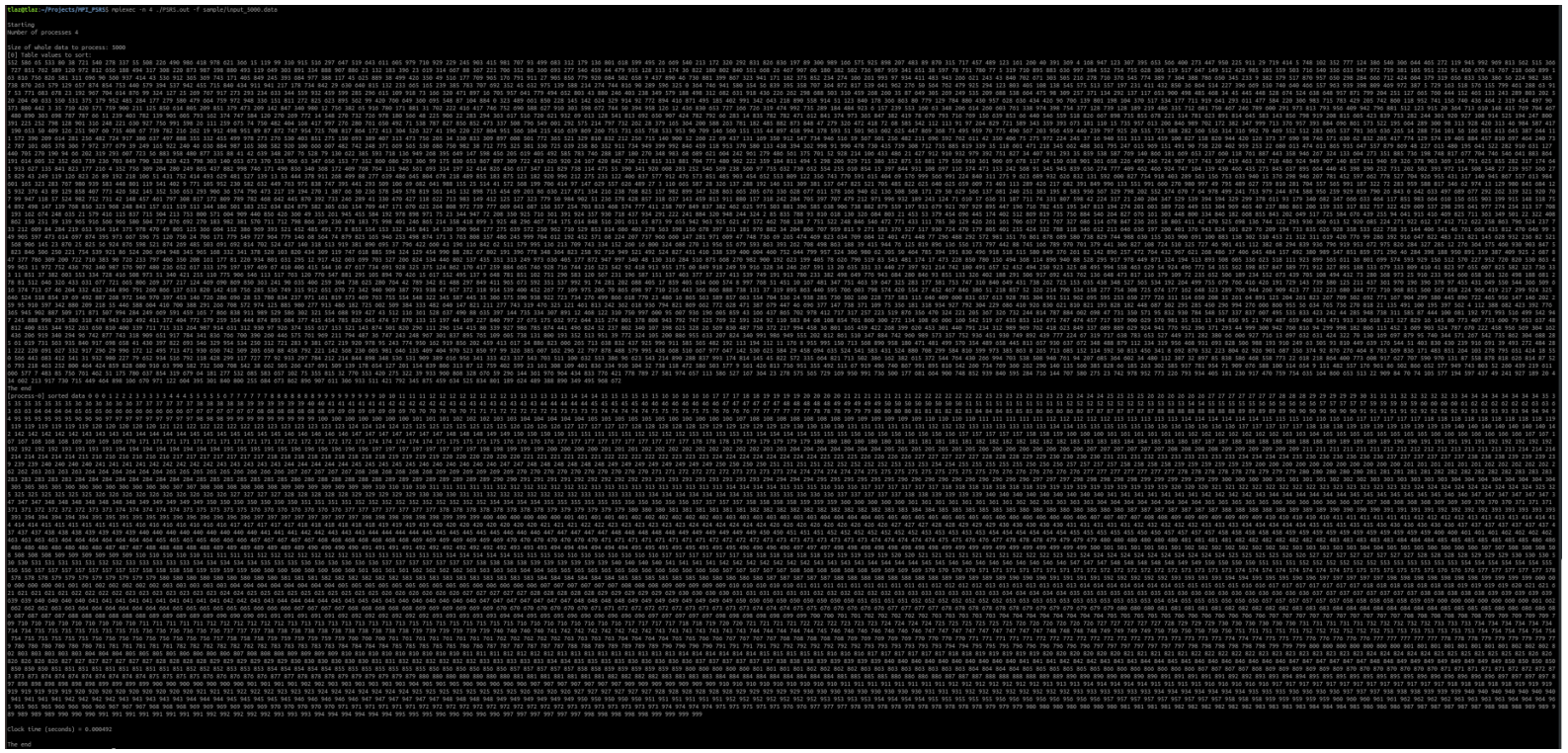
Na rysunkach rys. 4, rys. 5, rys. 6 oraz rys. 7 niektóre komentarze zostały wyłączone dla większej ilości danych w celu zwiększenia czytelności.



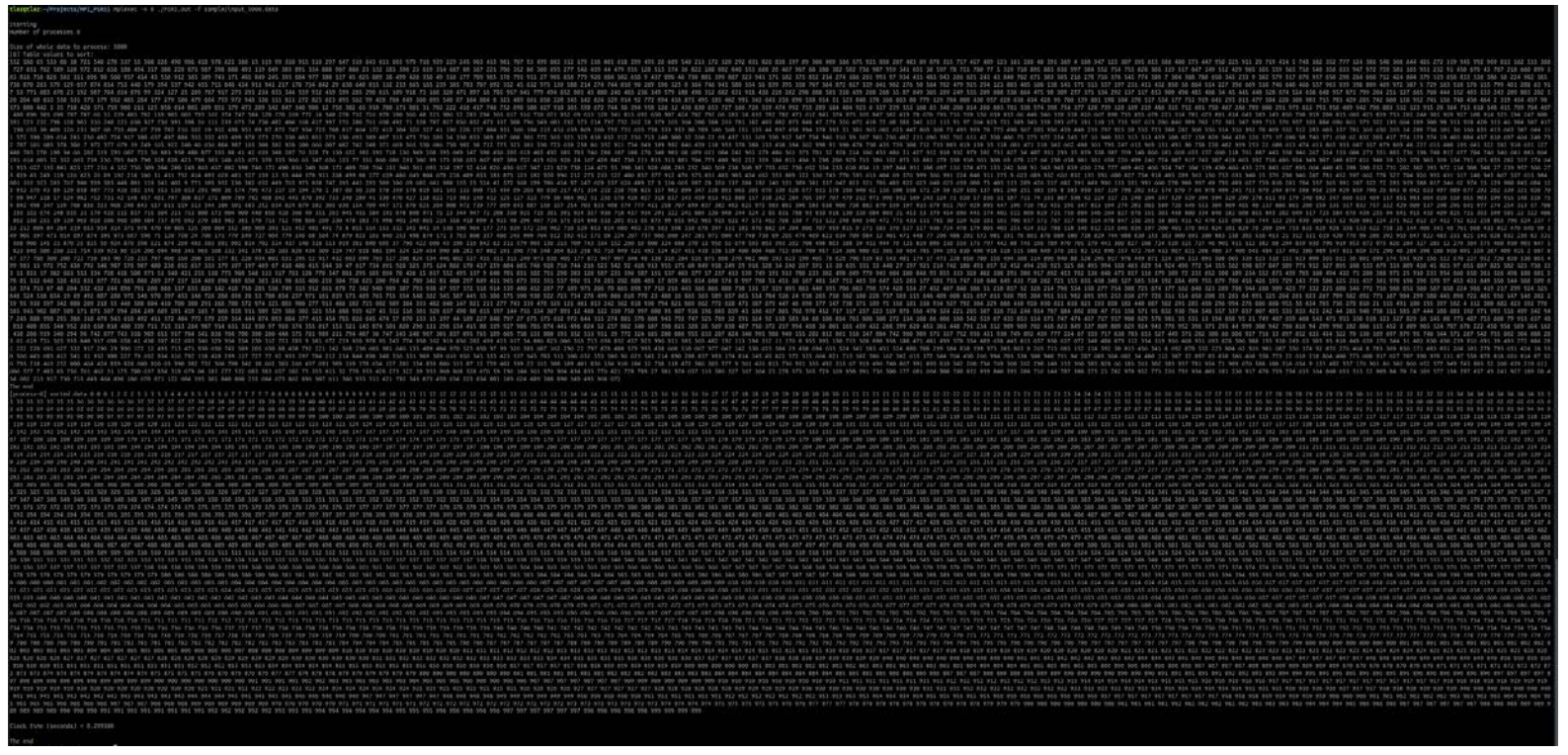
Rys. 4 Przykładowe wyjścia z programu dla N=4 FILE=sample/input_500.data



Rys. 5 Przykładowe wyjścia z programu dla N=8 FILE=sample/input_500.data



Rys. 6 Przykładowe wyjścia z programu dla N=4 FILE=sample/input_5000.data



Rys. 7 Przykładowe wyjścia z programu dla N=8 FILE=sample/input_5000.data

Na rysunkach rys. 4, rys. 5, rys. 6 oraz rys. 7 najważniejszy jest czas wykonania programu, ilość węzłów/procesów.

5. Źródła

Quinn, zad. 15.5, Grama, Gupta, rozdz. 13
http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/html/PSRSalgorithm.html
http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/html/PSRSAppendixI.html