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

1. 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.

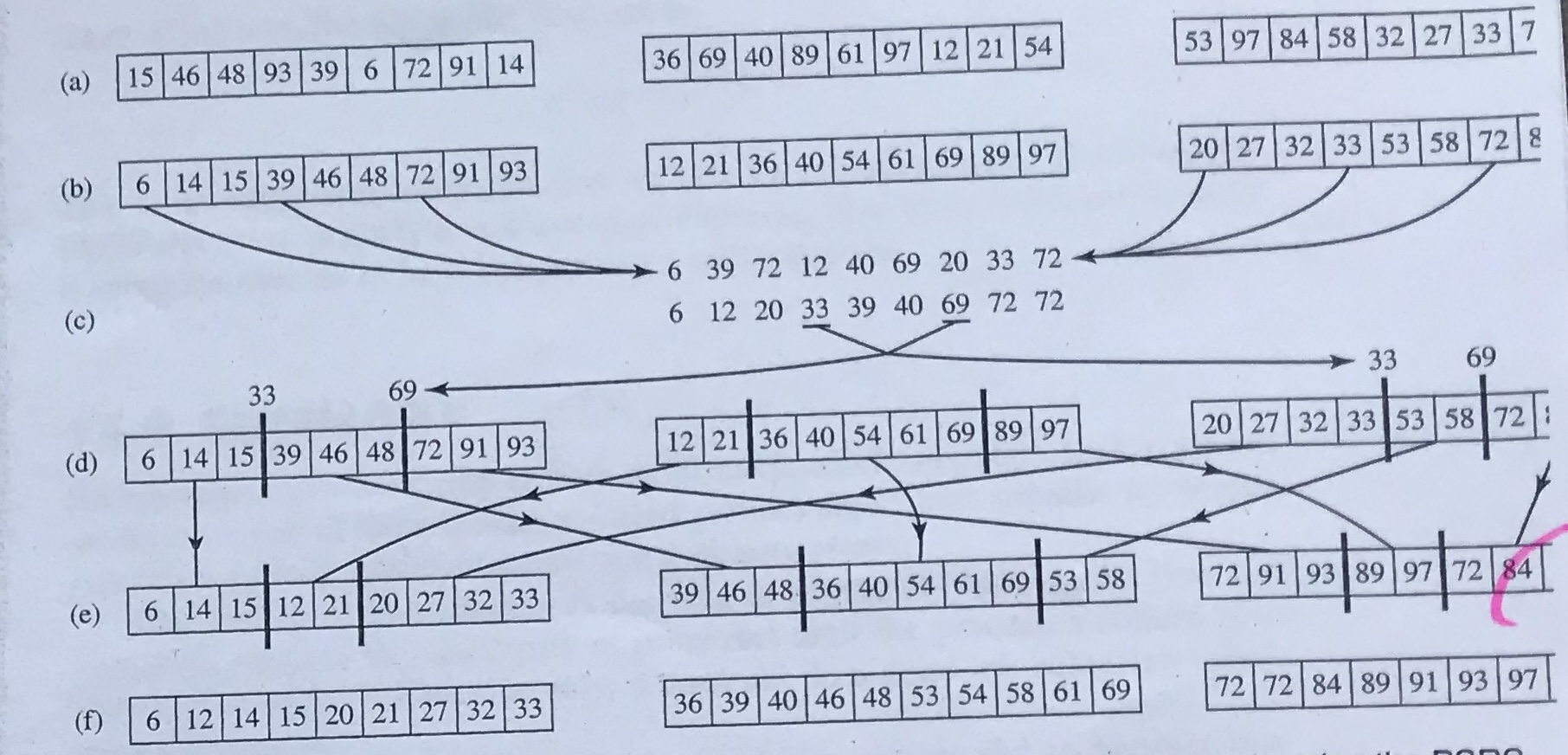
1. Faza III

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

1. 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.

1. 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

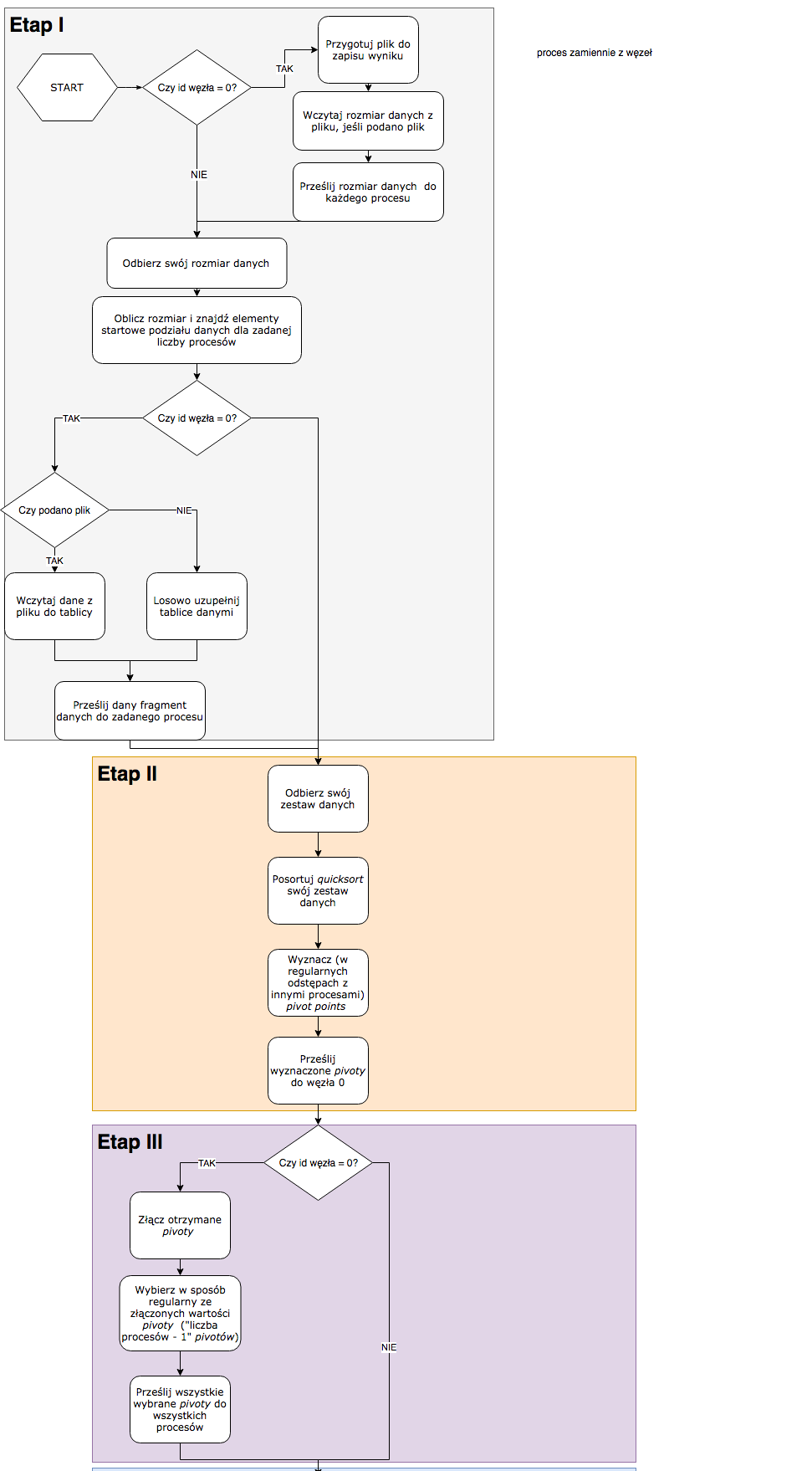
W projekcie znajduje się również katalog doc z dokumentacją.

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

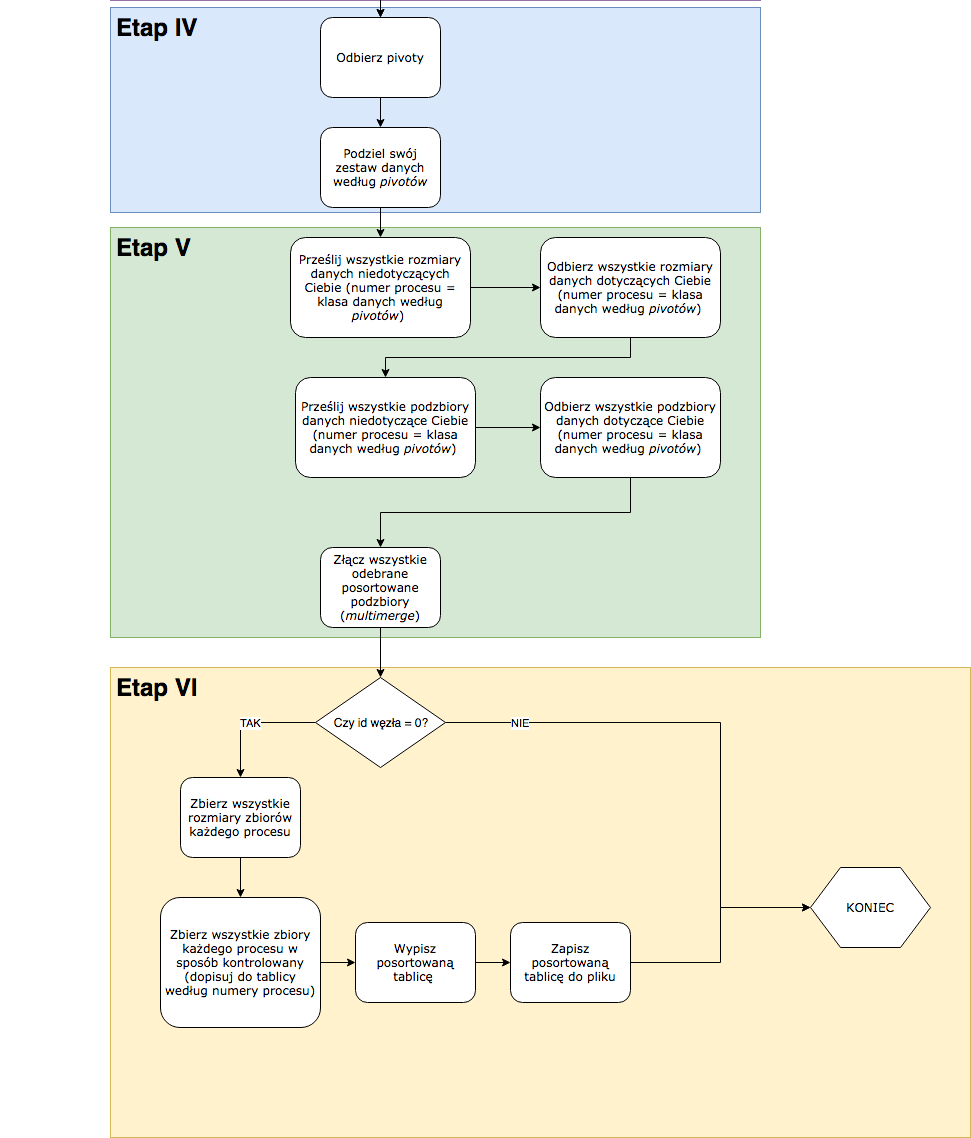
1. Opis działania

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

Diagram został również umieszczony w folderze doc/



Diag. 1



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 pivotysą 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

Diag.2

1. ETAP 5 - W następnym kroku każdy proces pozyskuje tylko dane z zadanej przez identyfikator partycji od każdego procesu i je łączy.
2. ETAP 6 - Na końcu dane te są przesyłane do serwera, który je łączy i wypisuje do pliku
3. Opis obsługi programu

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

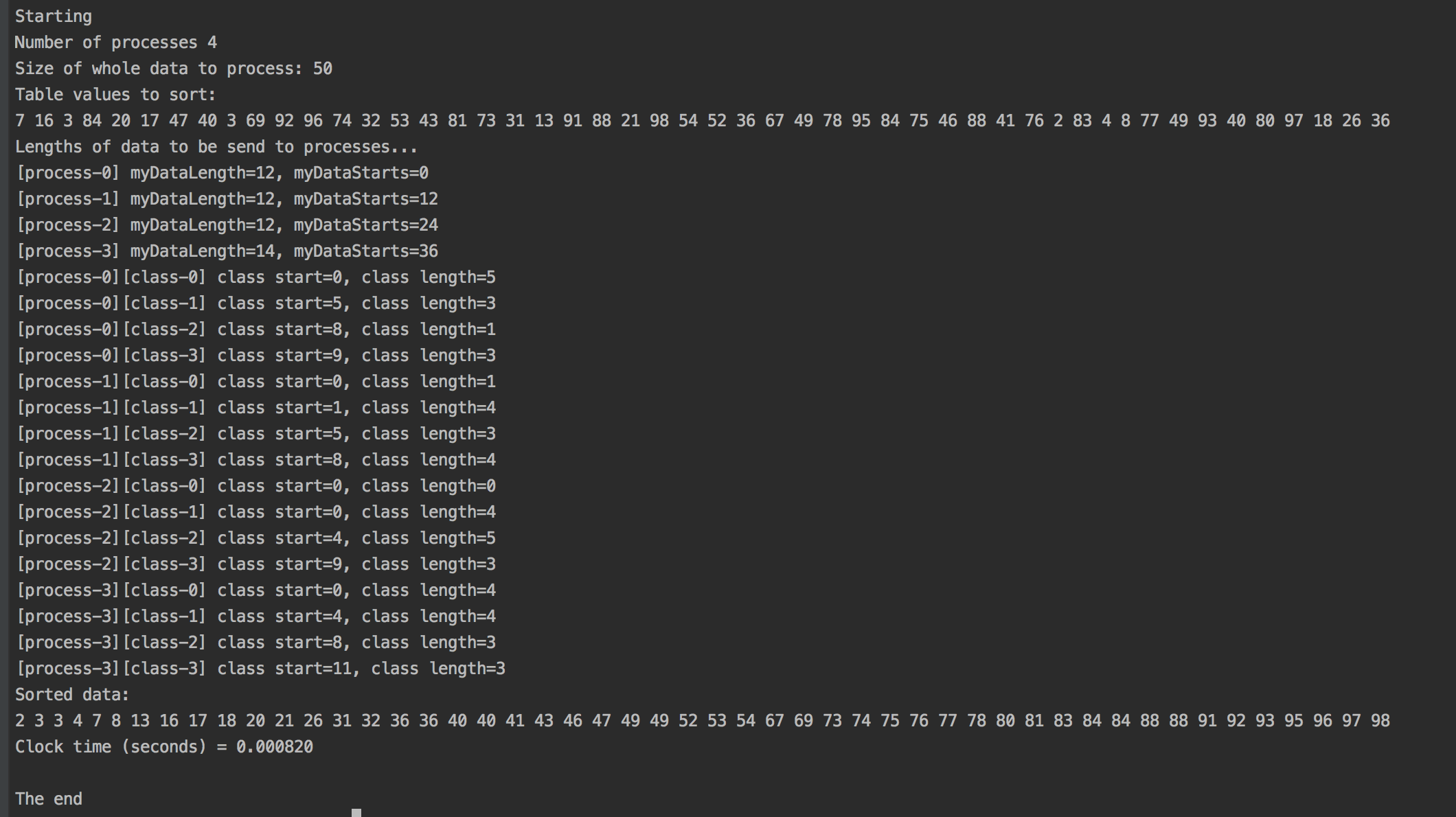
Udostępnia on 4 reguły:

* + *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 przedstawiono wyjście z programu dla danych z pliku przekazywanego jako parametr make’a oraz dla 4 procesów.

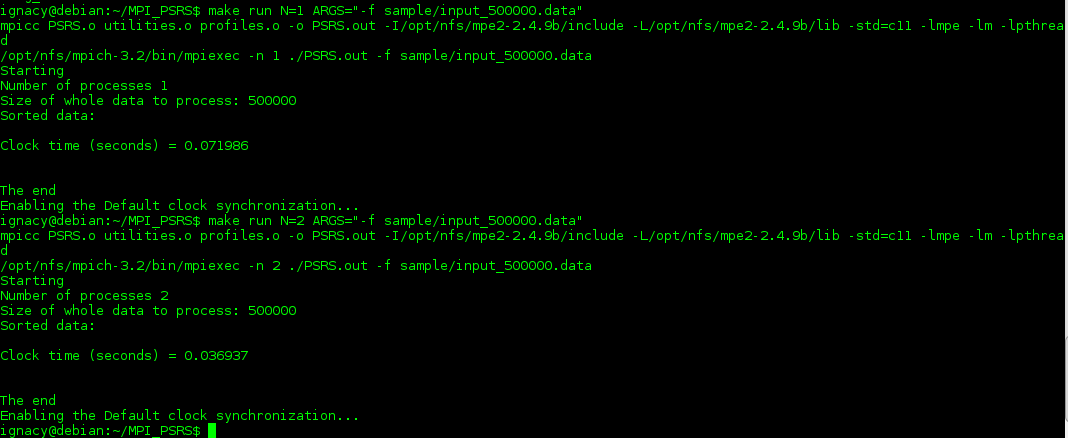


Rys. 2 Przykładowe wyjście z

make run N=4 FILE==sample/input\_50.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
* podział danych na danym procesie ze względu na pivoty wraz z długością danych w danej partycji
* posortowane dane
* czas działania algorytmu

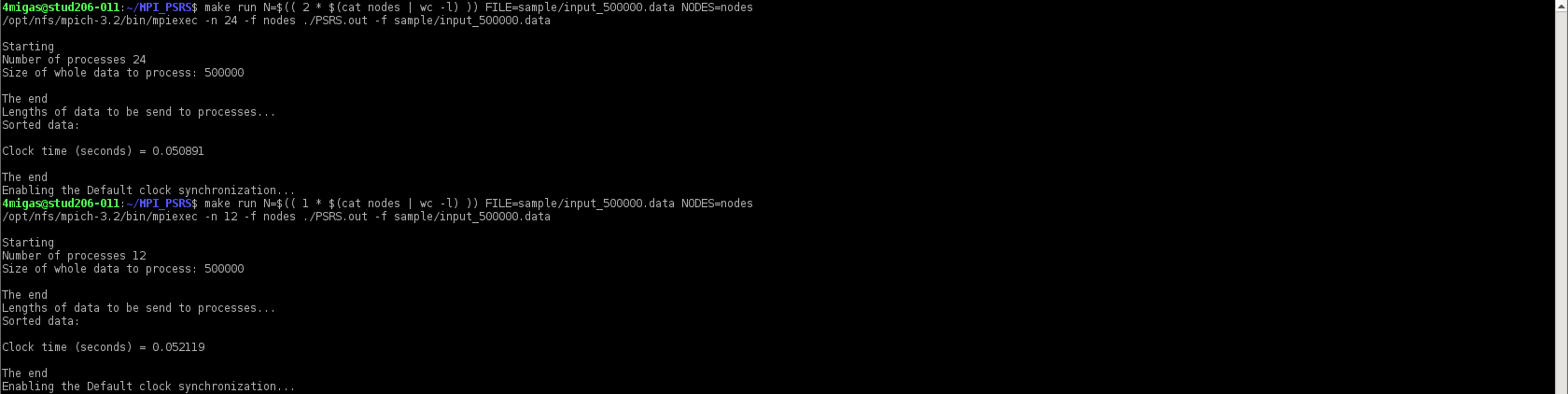
Na rysunkach rys. 3, rys. 4 oraz rys. 5 niektóre komentarze zostały wyłączone dla większej ilości danych w celu zwiększenia czytelności.

Rys. 3 Przykładowe wyjścia z programu



Rys. 4 Przykładowe wyjścia z programu

Rys. 5 Przykładowe wyjścia z programu

Na rysunkach rys. 3, rys. 4 oraz rys. 5 najważniejszy jest czas wykonania programu, ilość węzłów/procesów.

1. 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/PSRSalgorithm.html)

*http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/html/PSRSAppendixI.html*