Tomasz Łaz, Ignacy Migas

dr inż. Dydejczyk

6 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.

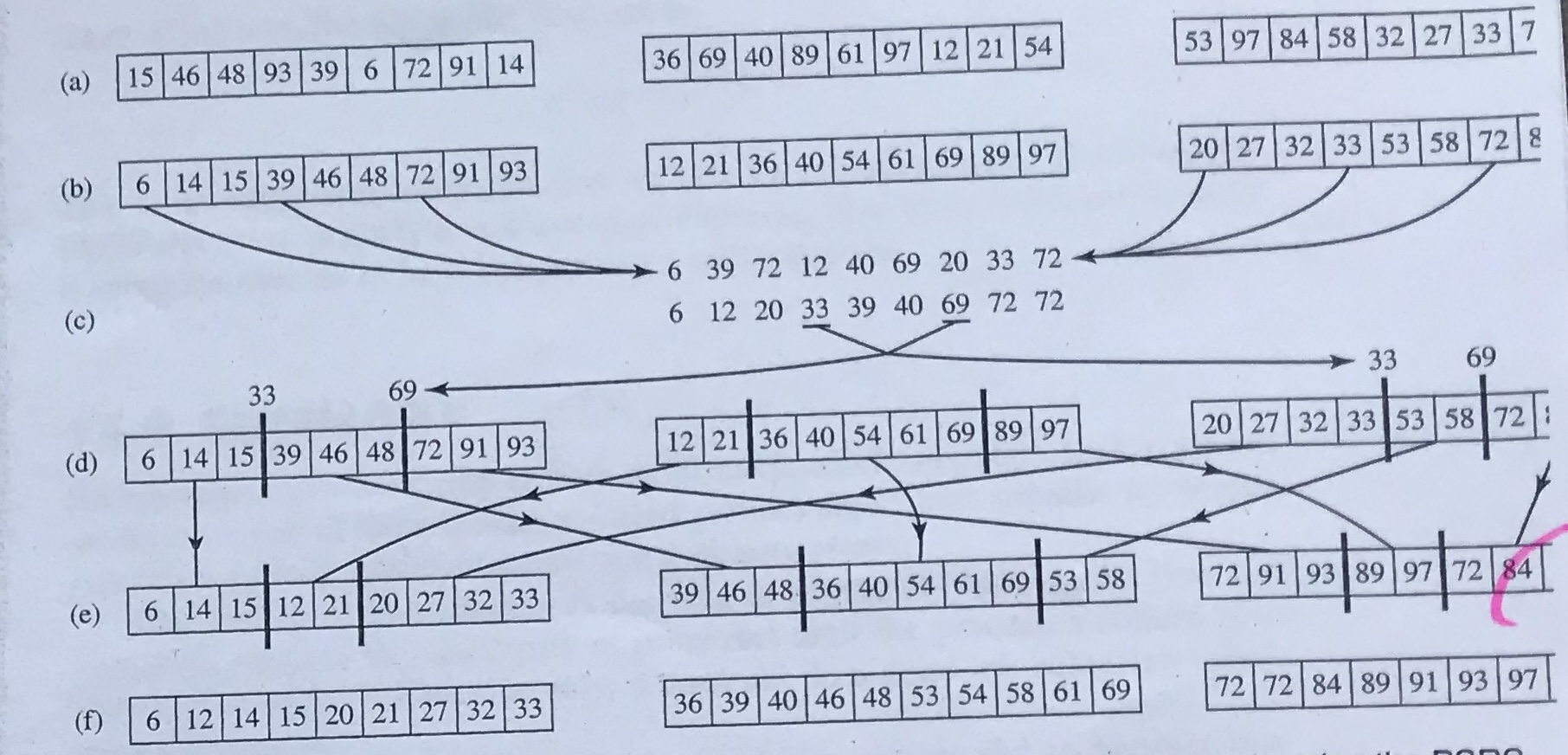
1. Faza III

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 5 plików:

* utilities.h + utilities.c
* profiles.h + profiles.c
* PSRC.c

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

W plikach profiles występują metody MPI, nadpisane, tak aby zrealizować profilowanie.

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

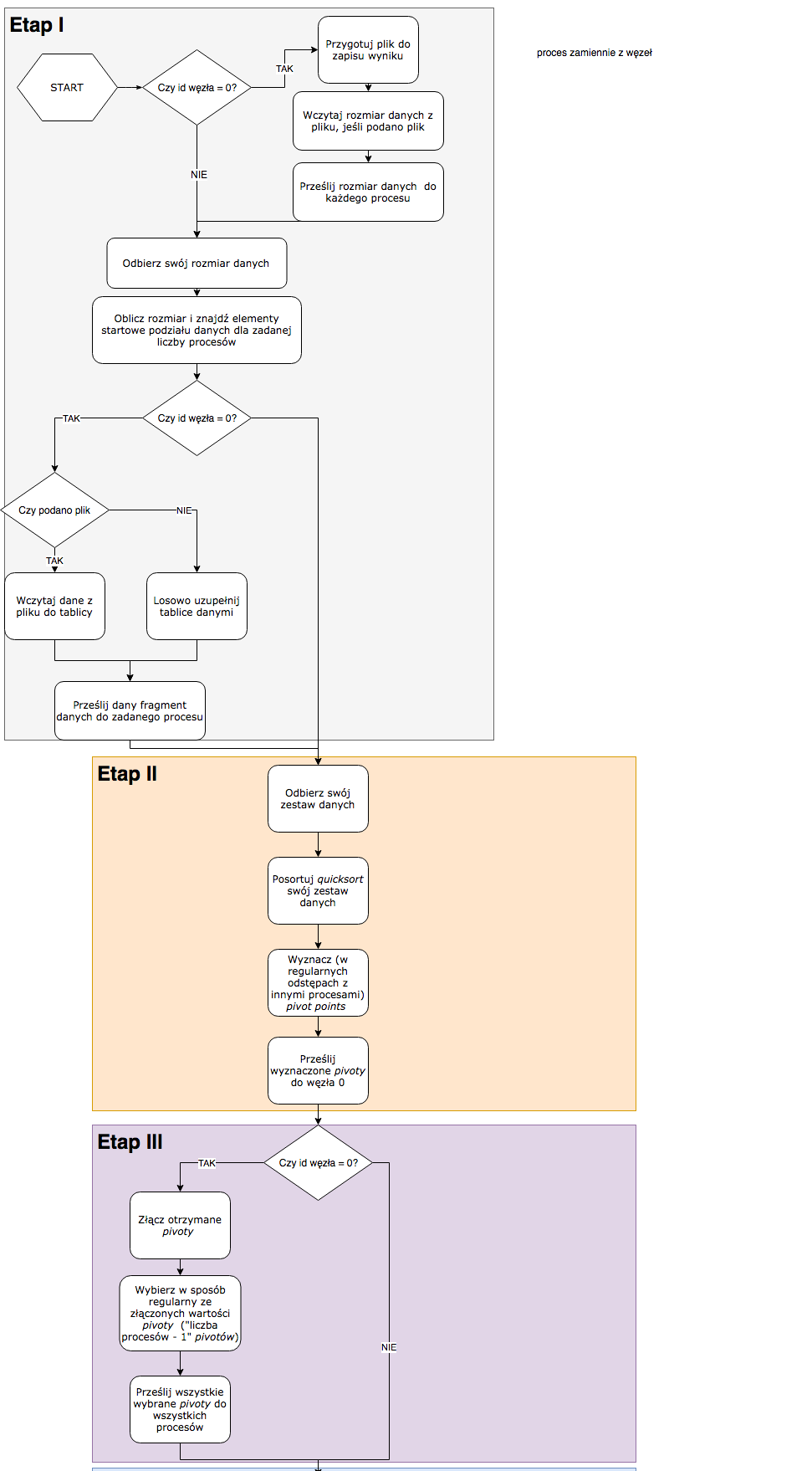
W projekcie znajduje się również katalog doc z dokumentacją.

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

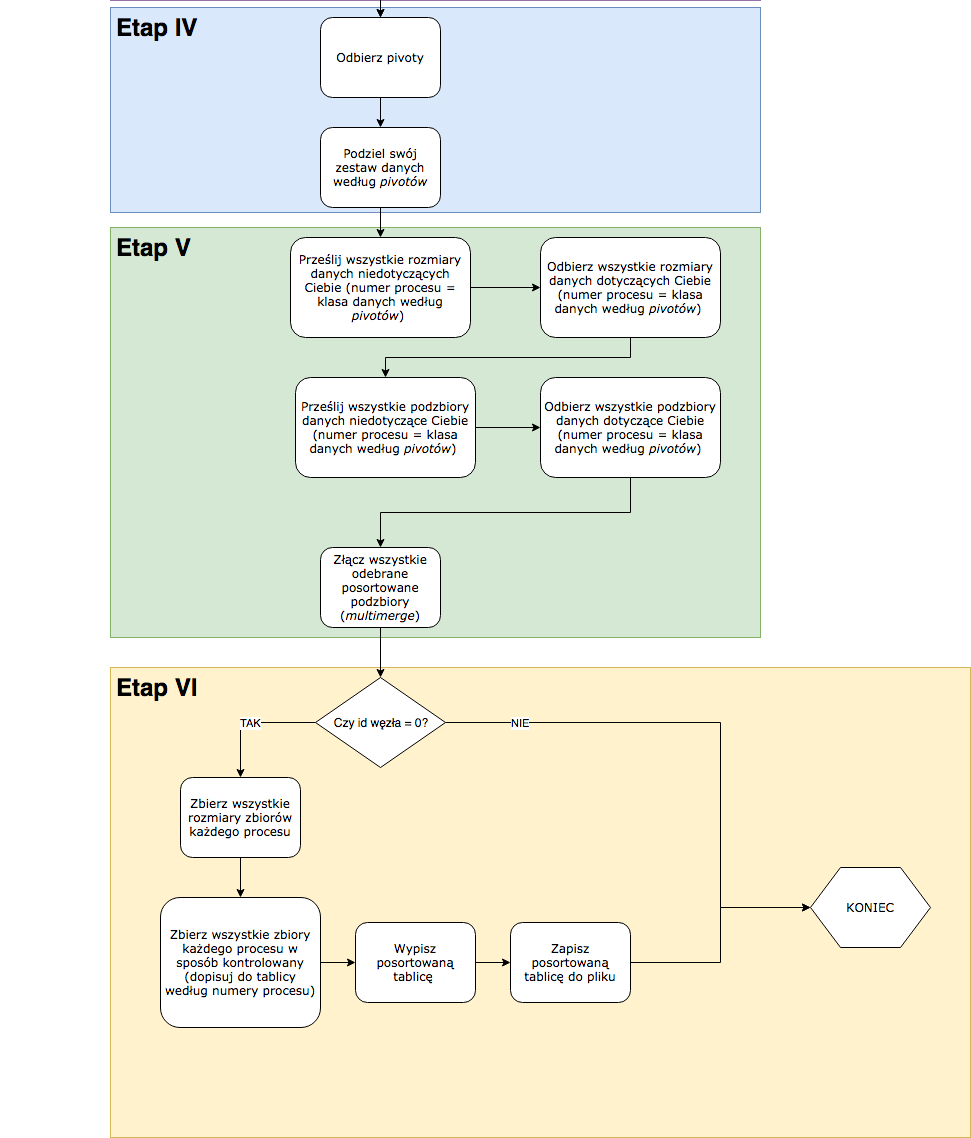
1. Opis działania

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

Diagram został również umieszczony w folderze doc/

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

Diag. 1

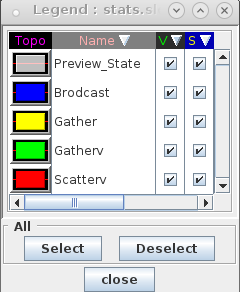
1. ETAP1 - Początkowo aplikacja przygotowuje dane do posortowania
2. ETAP 2 - W kolejnej fazie, dane są rozdzielane na procesu i wybierane są wartości, będące *pivotami*
3. ETAP 3 *-* Następnie pivotysą przesłane do servera, który je łączy i wybiera „liczba procesów - 1” pivotów. Pivoty te są przesyłane 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

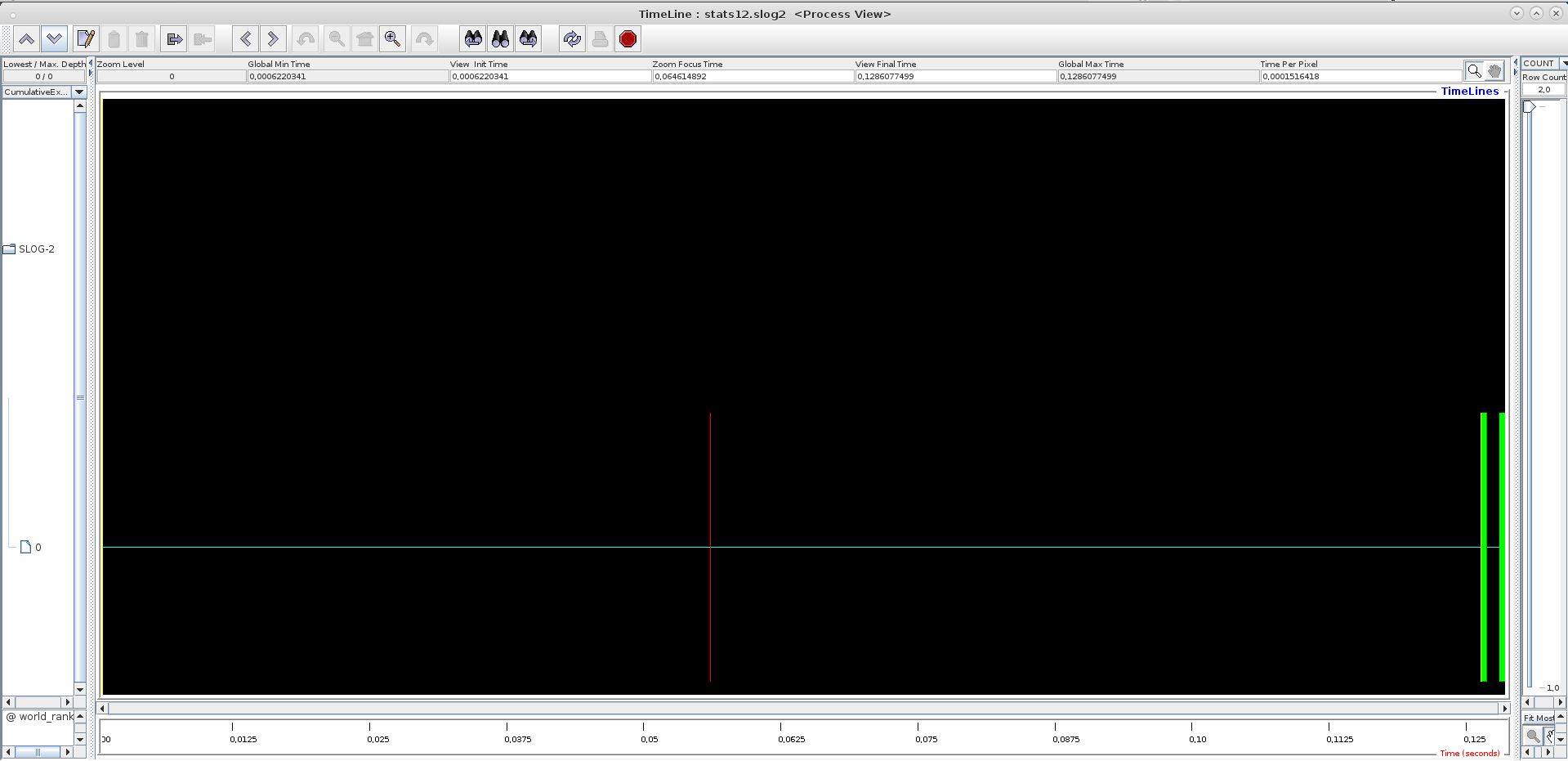
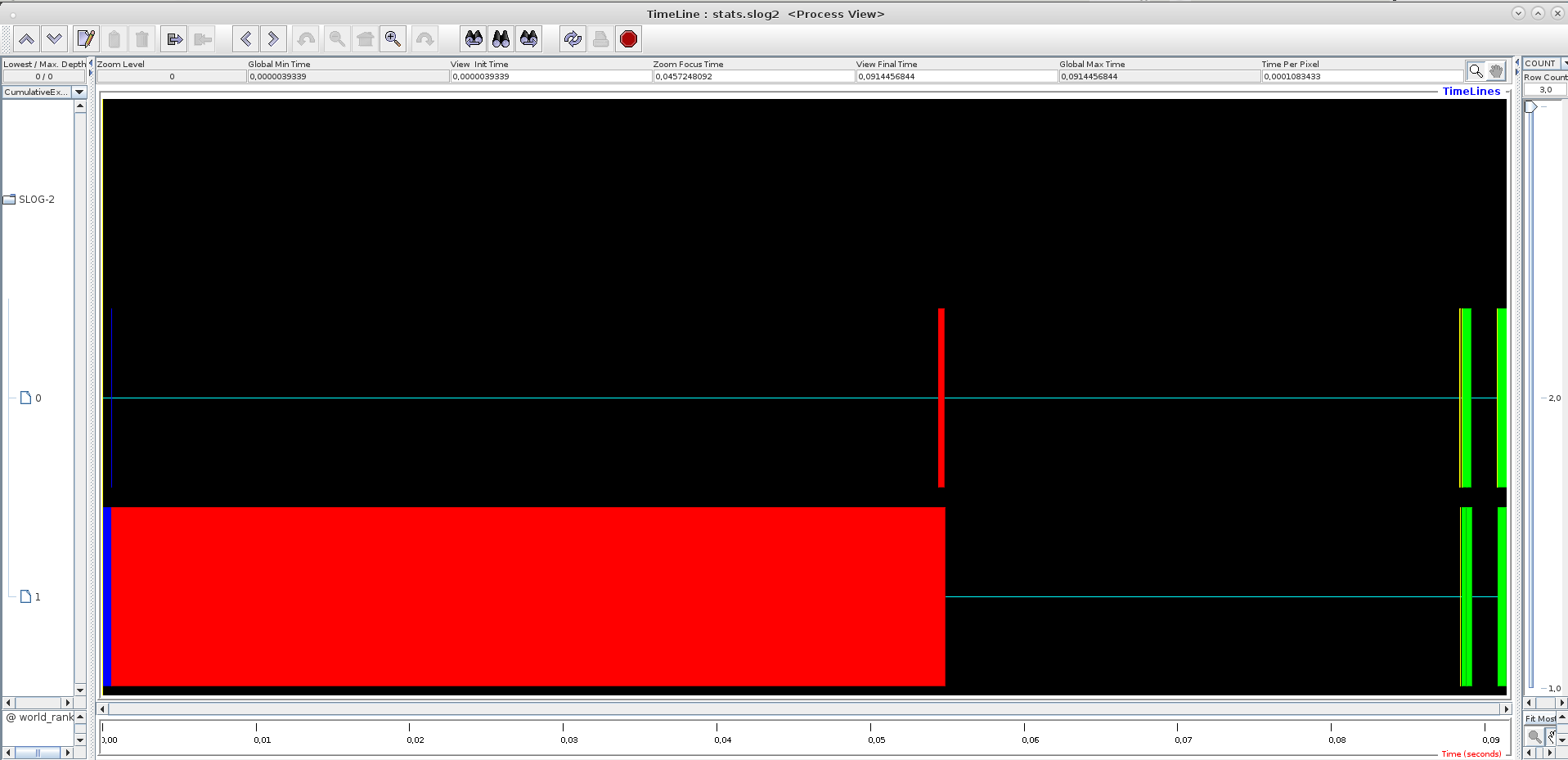
Kroki do skompilowania i uruchomienia na jednym węźle:

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. Profilowanie

W ramach projektu zrealizowano profilowanie

Kolory wykorzystane w wykresach przedstawia grafika Rys 2:

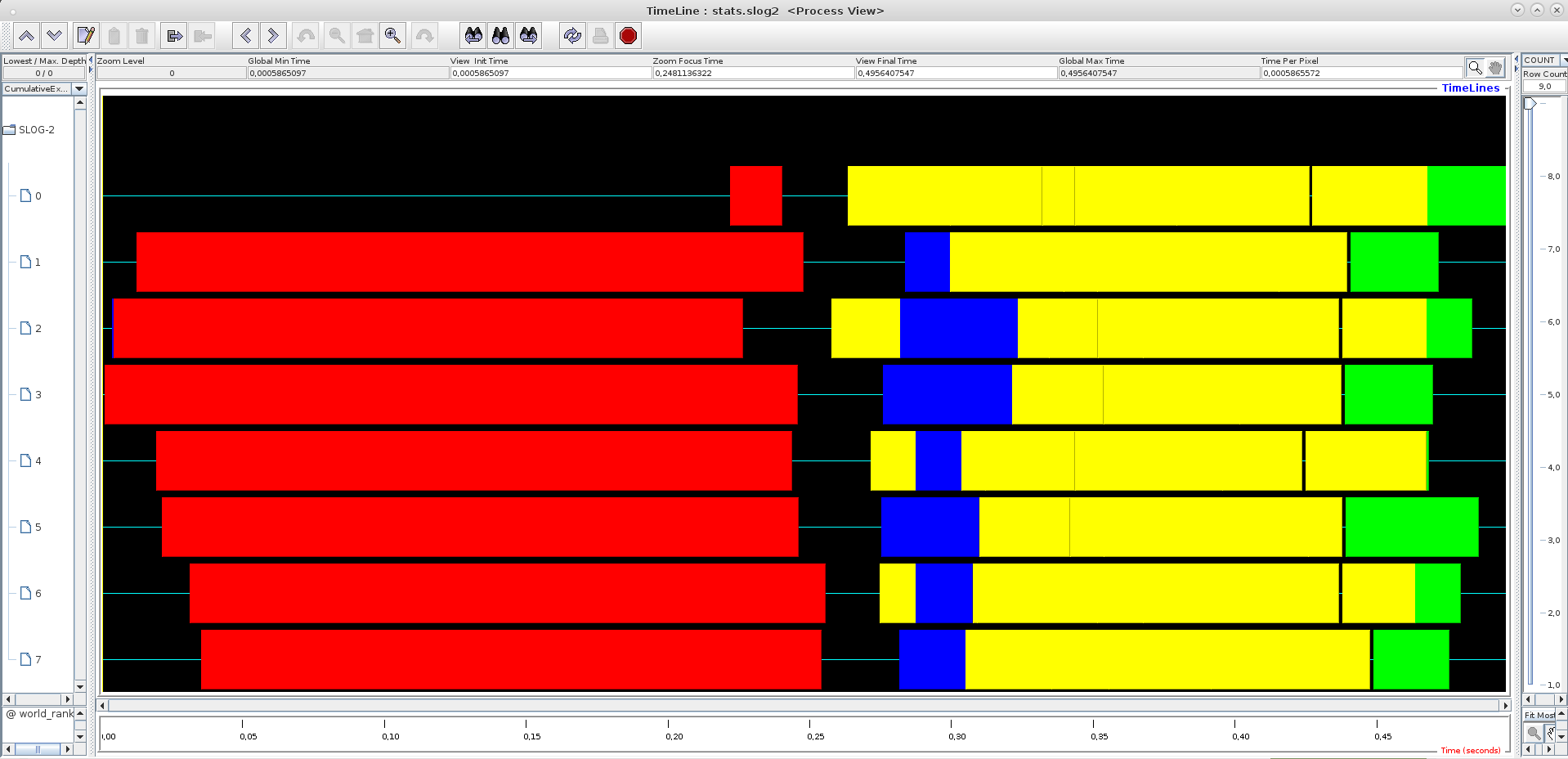
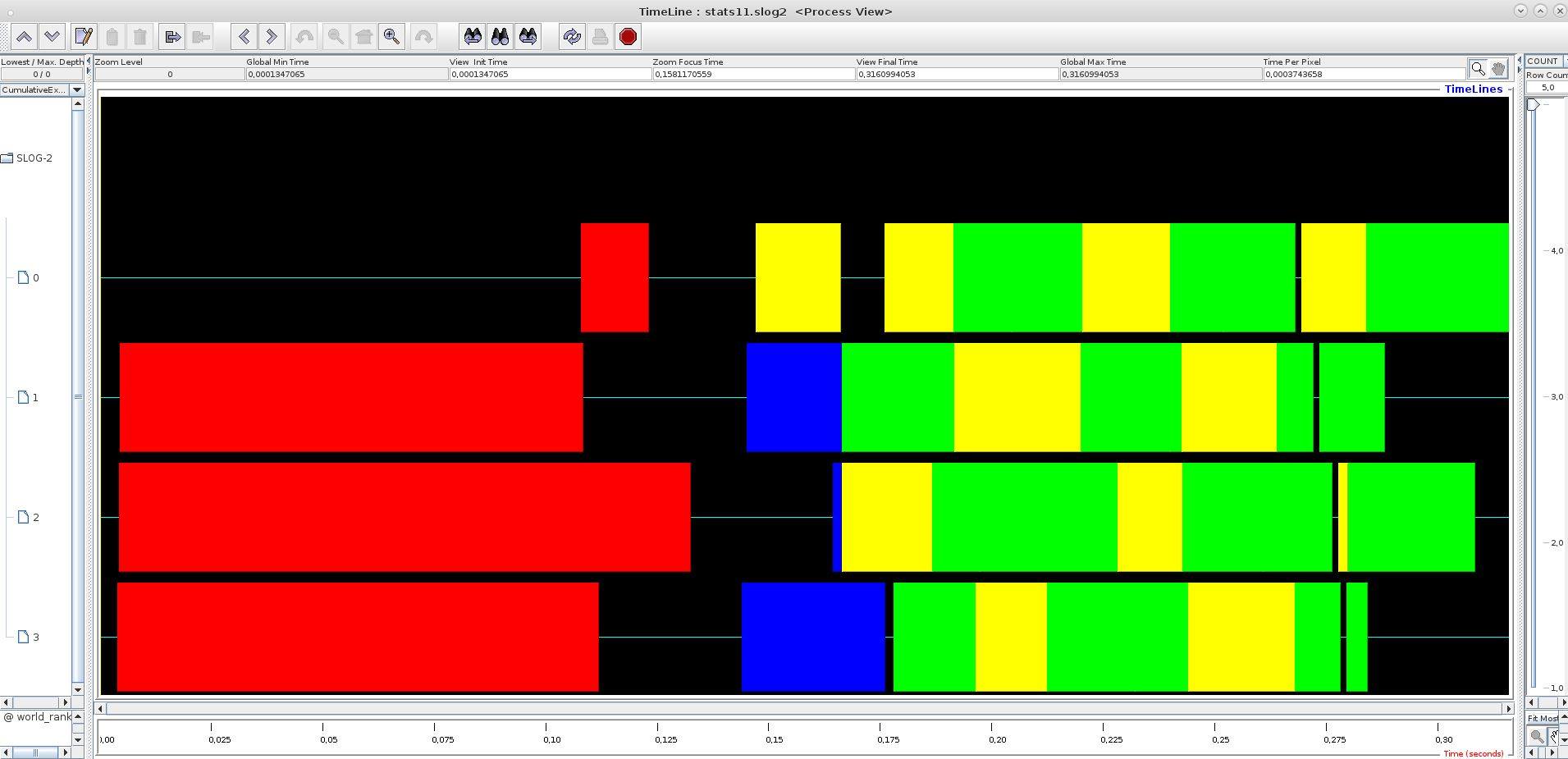
Rys. 2 Funkcje i ich kolory

Profilowanie zrealizowano na tej samej ilości danych dla 1, 2, 4 i 8 procesów co przedstawiono na rysunkach Rys. 3, Rys. 4, Rys. 5, Rys. 6

Rys. 4 Wykres czasowy aplikacji dla 2 procesów

Rys.3 Wykres czasowy aplikacji dla 1 procesu

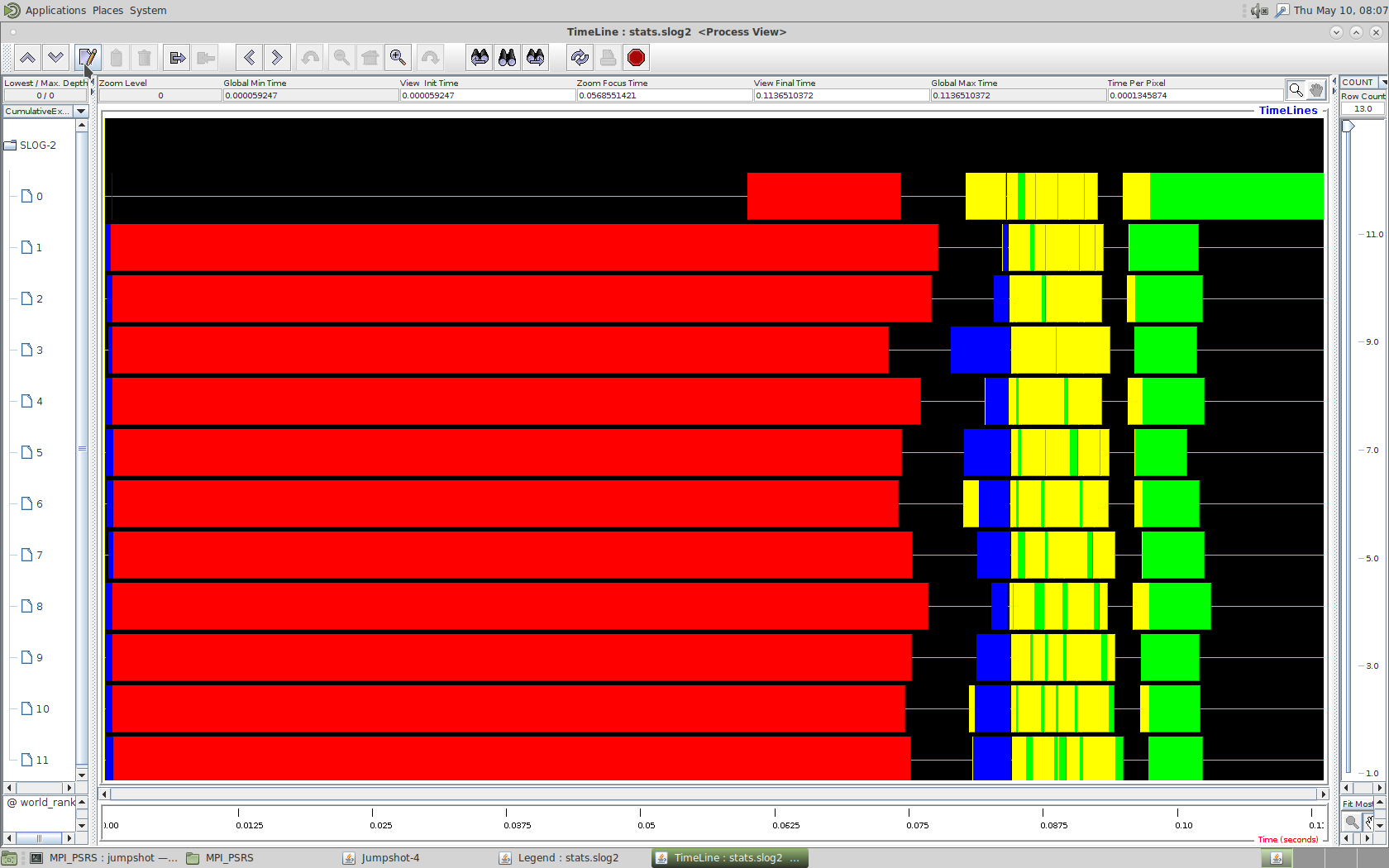
Na rysunkach 3,4, 5 oraz 6 można zaobserwować, że najlepszy wynik został osiągnięty dla liczby procesów równej 2. Wynika to z najbardziej optymalnego zestawu ilości przesyłań komunikatów pomiędzy procesami, a ilością rdzeni procesora.

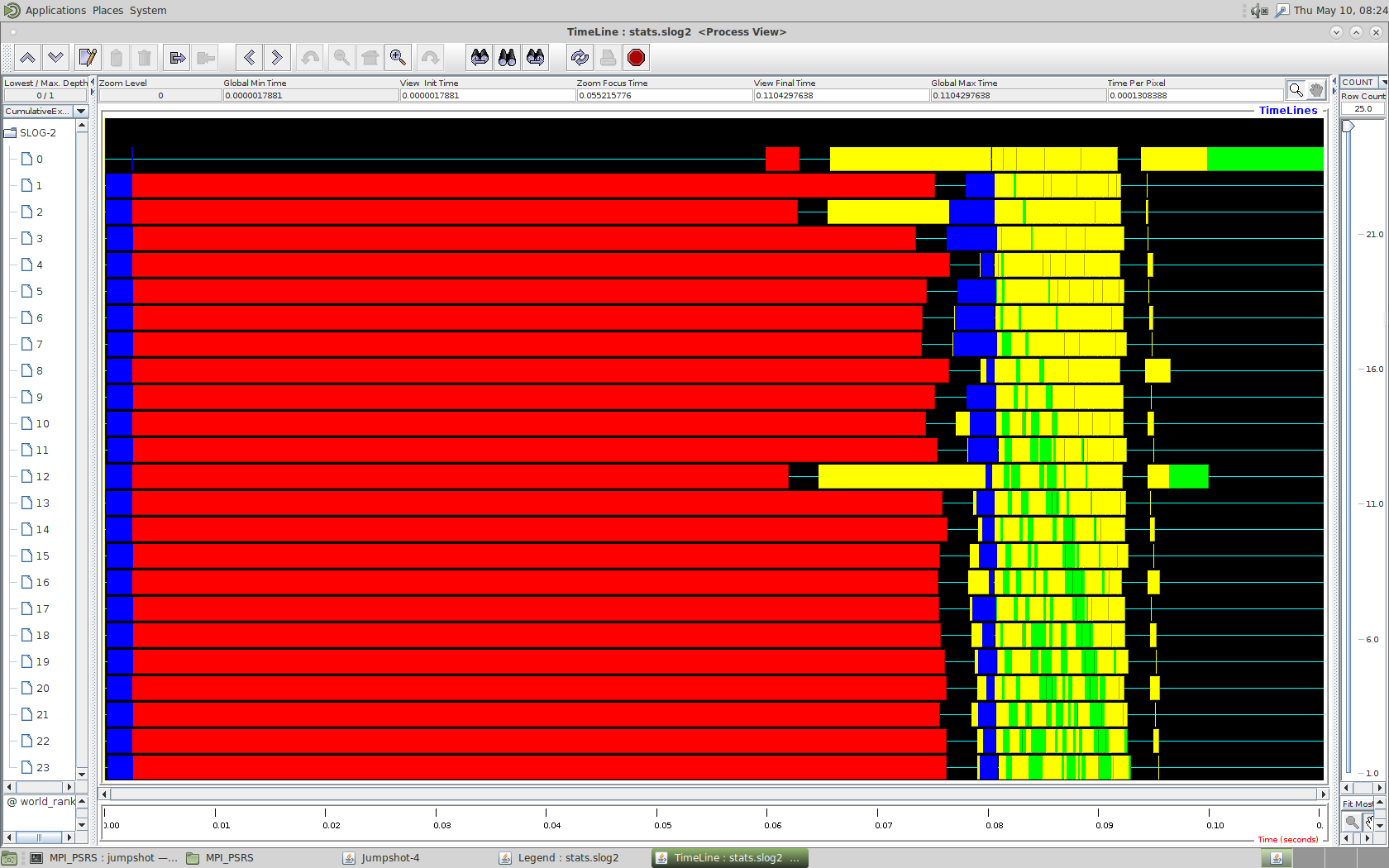
Dla jednego procesu narzutu komunikacyjnego nie obserwujemy, ale samo sortowanie trwa długo. Z kolei dla 8 procesów same obliczenia trwają krótko, jednak komunikacja pomiędzy wszystkimi procesami trwa na tyle długo, że wynik jest gorszy niż sortowanie lokalnie. Podobnie w przypadku 4 procesów.

Rys. 5 Wykres czasowy aplikacji dla 4 procesów

Rys. 6 Wykres czasowy aplikacji dla 8 procesów

Rys. 7 Wykres czasowy aplikacji dla 12 procesów uruchomionych na 12 węzłach

Zrealizowano profilowanie również dla kilku węzłów, co przedstawiono na rysunkach Rys. 7, Rys. 8



Odpowiedni dobór ilości procesów/węzłów dla zadania zależy od ilości danych do posortowania, specyfikacji sprzętu i znajomości infrastruktury klastra.

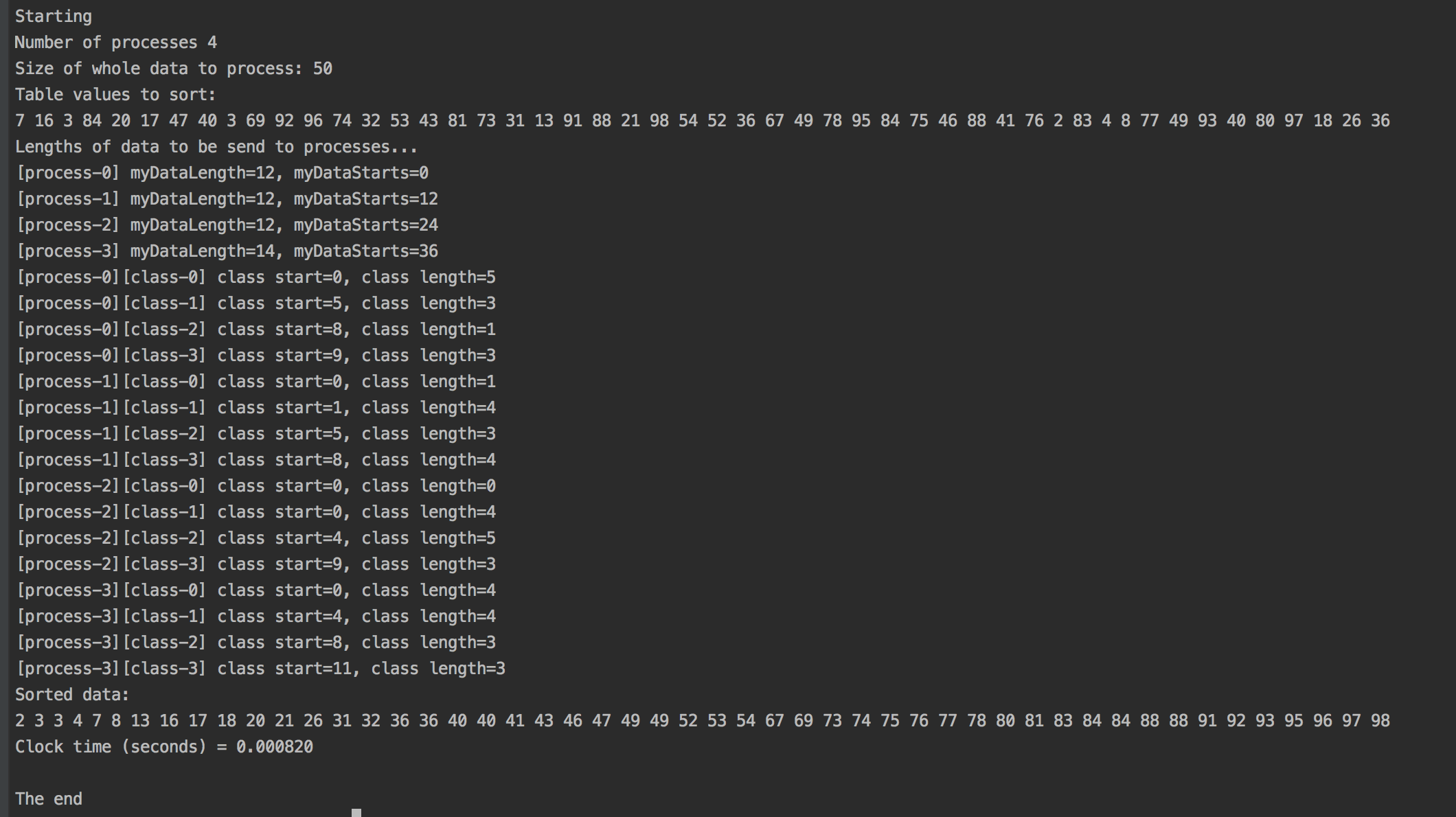
Problem sortowania jest wrażliwy i wybierając odpowiednie parametry należy pamiętać o wszystkich elementach, które mogą opóźniać proces sortowania równoległego. Niejednokrotnie może okazać się korzystniej sortować dane lokalnie na jednym węźle niż próbować takie sortowanie zrównoleglać.

Rys. 8 Wykres czasowy aplikacji dla 24 procesów na 12 węzłach

Dodatkowe wyniki zostały przedstawione w rozdziale 5.

1. Przykład wyjścia z programu

Na rys. 9 przedstawiono wyjście z programu dla danych z pliku przekazywanego jako parametr make’a oraz dla 4 procesów.

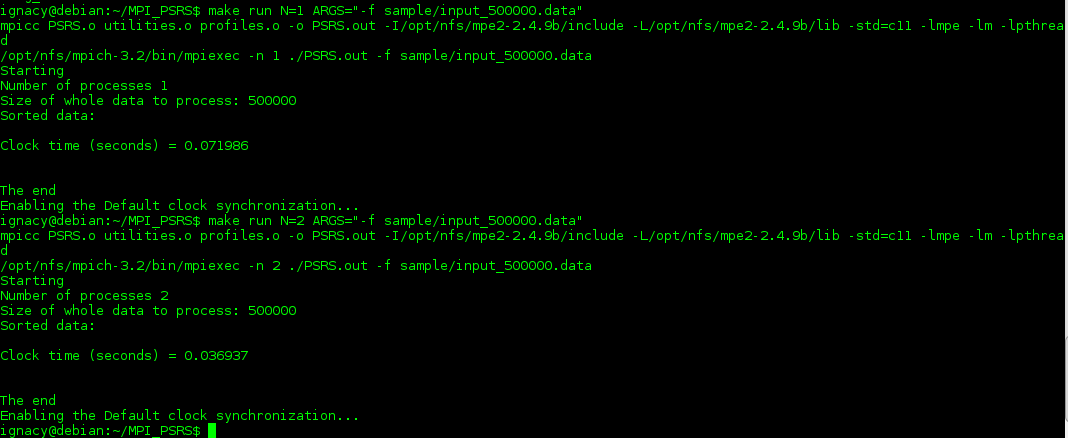


Rys. 9 Przykładowe wyjście z

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

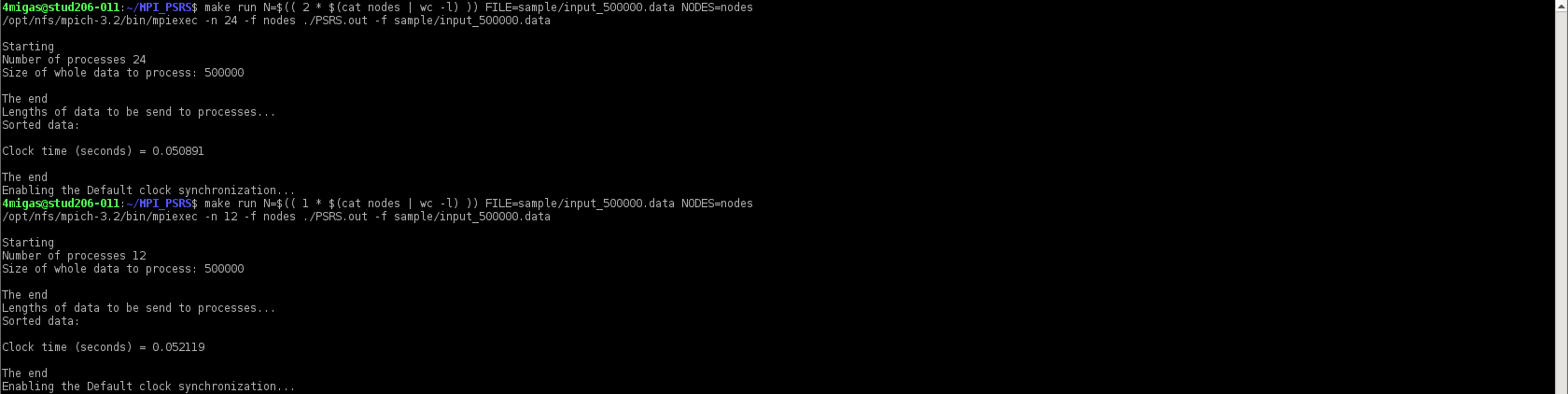
Na rysunkach rys. 10, rys. 11 oraz rys. 12 niektóre komentarze zostały wyłączone dla większej ilości danych w celu zwiększenia czytelności.

Rys. 10 Przykładowe wyjścia z programu



Rys. 11 Przykładowe wyjścia z programu

Rys. 12 Przykładowe wyjścia z programu

Na rysunkach rys. 10, rys. 11 oraz rys. 12 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/PSRSAppendixI.html*