Praca domowa 9 – hypercube

Michał Szczygieł

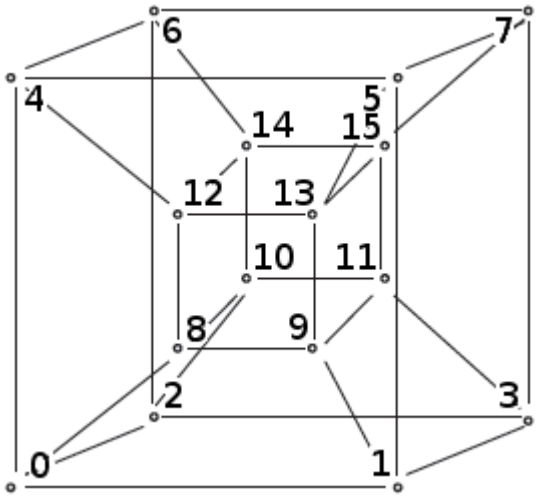
Wstęp

Zadanie składa się z 4 plików:

- Main.java
- Node.java
- Hypercube.java
- Algorithm.java

CEL:

Stworzenie architekury hiperześcianu i metod dystrybucji oraz redystrybucji danych wewnatrz tego hiperszescianu. Każdy wezeł jest reprezentacją równoległego procesora, wykonującym obliczenia na wczesniej otrzymanych danych.



Klasa Main

Zadaniem tej klasy jest wczytanie danych z pliku, oraz przekazanie tych wartości do klasy Algorithm. Również oprócz danych z pliku klasa ta winna przekazać parametr n-size (określający rozmiar hypercuba - ilość węzłów), który również jak i parametr file path nadawana jest jako argument wywoławczy programu.

Metody:

```
public static void main(final String... args) throws FileNotFoundException
```

- Metoda główna, przyjmuje dwa parametry filePath i n-size, przekazuje wczytane dane i rozmiar do klasy Algotithm, wykonując cel, oraz wyświetlenie wyniku sumy dla pobranych danych z dokładnością do 5 miejsc dziesiętnych.

```
private static ArrayList<Double> readFile(File file)
            throws FileNotFoundException {
```
– Metoda ta wczytuje dane w postaci liczb rzeczywistych z podanej ścieżki oraz zwraca przefiltrowane dane.

Klasa Algorithm

Klasa otrzymawszy dane wywołuje w konstruktorze konstruktor klasy dziedziczonej, który buduje hiperszecian o zadanym rozmiarze (składający się z tylu wezłów co wartość parametru n-size) oraz ustawienia puli wątków, równoważenej ilości wezłów. Głównym zadaniem tej klasy są operacje artymetyczne (obliczanie sumy), jak i logika dystrybucji redystrybucji i komunikacji między węzłami.

Ważniejsze metody:

```
private void prepareDistrbuteData(List<Double> data, Integer current,
            Integer level)
```
– metoda ta wykonywana jest w konstrukotrze klasy Algorithm. Jej zadaniem jest podział danych na równe cześci i przesłanie ich do węzłów. Algorytm opisujący działanie tego mechanizmu jest opisany w nagłówku „Realizacja algorytmu w klasie Algorithm” jak i w kodzie źródłowym tej klasy.

```
private Double redistributeData(Double sum, Integer current, Integer level)
```
– metoda ta wykonuje operację odwrotna do dystrybicji a mianowicie, przekazuje obliczoną sumę do sąsiadujących węzłów. Algorytm opisujący działanie tego mechanizmu jest opisany w nagłówku „Realizacja algorytmu w klasie Algorithm” jak i w kodzie źródłowym tej klasy.

Klasa Hypercube

Powyższa klasa tworzy architekturę hiperześcianu. Posiada ona również metodę publiczną run która uruchamia proces w całej strukturze.

```
public void run() throws InterruptedException {
    setResults(executor.invokeAll(nodeList));
    executor.shutdown();
}
```

Klasa ta również posiada dwie abstrakcyjne metody:

```
public abstract void put(Node node, List<Double> input);
public abstract Double get(Node node);
```

Klasa Node

Klasa ta reprezentuje pojedynczy węzeł architektury. Klasa ta również implementuje interfejs Callable. Po wywołaniu metody run w klasie Hypercube, wykonywana jest automatycznie metoda w każdym węźle:

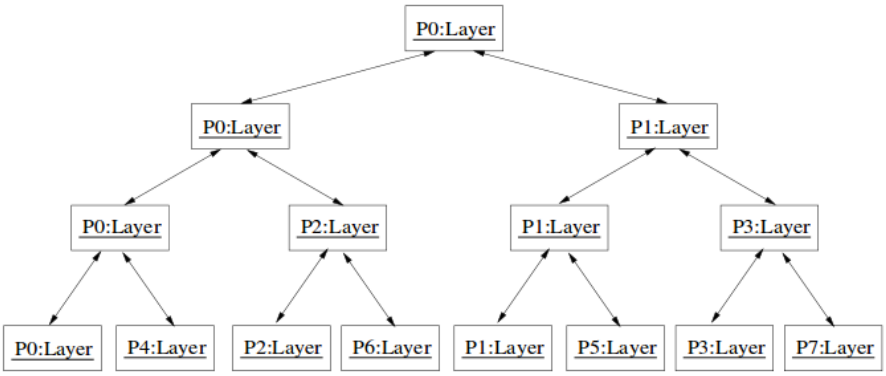
```
public Double call() throws Exception {
    setLocalSum(Algorithm.sum(getData()));
    setIsDone(true);
    return getLocalSum();
}
```

Realizacja algorytmu w klasie Algorithm

Bazowa komunikacja w hupersześcianie odbywa się według poniższego algorytmu:

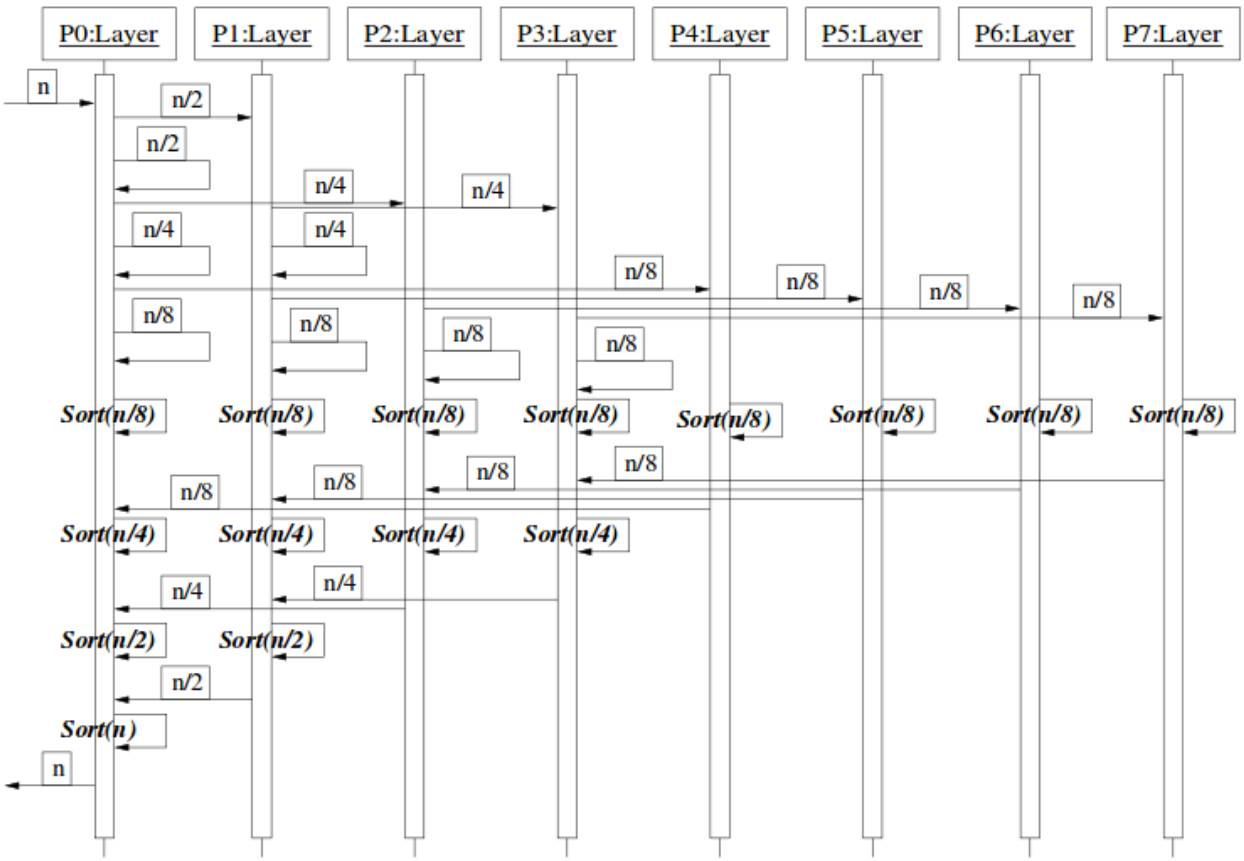
```
procedure hypercube(myid, input, logp, output)
begin
  state = input                    ! Local state = input
  for i = 0 to logp-1              ! Repeat logp times
    dest = myid XOR 2^i            ! Determine comm partner
    send state to dest             ! Exchange data
    receive message from dest      !
    state = OP(state,message)      ! Perform operation
  endfor                           !
  output = state                   ! Output = final state
end
```

Natomiast, problem ten rozwiązałem stosując drzewo binarne, ponieważ zaobserwowałem pewną zachodzącą zależność. W celu uniknięcia losowej propagacji danych, wynikającą ze wcześniejszego skończenia obliczeń, podzieliłem sześcian na następujące drzewo:



Każdy poziom drzewa może być podzielony na sąsiada prawego i lewego gdzie prawy jest potomkiem rodzica. Zachodząca zależność:z węzła bieżącego można przekazać podzieloną wartości do jednego sąsiada za jedną operacją i do samego sobie w celu kolejnej propagacji. Czyli lewa gałąź ma wartość current, a prawy $current + 2^{level-1}$. Poniżej zamieszczam schemat sekwencyjny działania hipercuba:

- gdzie n oznacza dane
- a Sort na rysunku oznacza operację sumowania.



Złożoność obliczeniowa dla tego programu wynosi:

- Rozwiązanie problemu sumowania dla danego wezła – $O(k)$, dla k*N liczb rzeczywistych, gdzie N ilość węzłów.

Dyskusja

Powyżej zaprezentowane rozwiązanie dystrybucji danych, wydaje się jednym z najbardziej optymlanych rozwiązań, zakładając jedynie że zbiór wejściowy jest znany. Operacja przy strumienu danych, mogłaby być zrealizowana na zasadzie stworzenia bufora, który by zbierał cześć danych a następnie przeprowadzał dystrybucję danych.