Курсов проект

по

Системи за паралелна обработка

Тема:

„Алгоритъм на Хъфман за компресия на данни – честотни таблици“

Изготвил:

Яна Руменова Георгиева, Ф.Н. 81281, Компютърни науки, трети курс, първи поток, трета група

Ръководител:

ас. Христо Христов

1. Идея на проекта

Целта на проекта е реализация на алгоритъма на Хъфман. Ще бъдат изследвани ускорението S_p, ефективността E_p и бързодействието на програмата T_p при различен брой нишки. Строенето на честотната таблица и четенето на входните данни ще бъдат паралелизирани.

2. Основни моменти

- Решението е имплементирано на програмния език C++.
- Използвани са STL, thread, както и външна библиотека реализираща ThreadPool. За повече информация : https://github.com/progschj/ThreadPool.
- Реализирани са два вида балансиране: статично(циклично планиране) и динамично, както и има възможност за промяна на грануларността.
- Сложността на решението, което е предложено, по време O(nlogn), където n е броя на различните символи участващи във входното съобщение. Използвана е структурата данни priority_queue от STL оптимизираща процеса на избор „най-леки“ дървета.

3. Описание на алгоритъма и функциите

- `int main(int argc, char* argv[])`

В нея се обработват параметрите подадени от командния ред и се извиква функцията `distributeTasksCalculateTheFrequencyTable()` и `mainFunctionBuildTheTree()`. Също в нея се засича времето за цялостното изпълнение на програмата.

- `void distributeTasksCalculateTheFrequencyTable()`

В нея се прави декомпозицията по данни и съответно спрямо входните параметри се осъществява статично или динамично разпределяне с вариране на грануларността.

В краят се събират данните от всички частично конструирани таблици и се изгражда главната честотна таблица:

`unsigned frequencyTableOfInputSymbols[MAX_SYMBOLS].`

- `void calculateFrequencyTableThreadI(const int& currStart, const int& length, const int& threadNumber)`

В нея нишката с номер I обработва съответното парче от входни данни като конструира своята частична честотна таблица:

```
unsigned
frequencyTableOfInputSymbolsThreadI[MAX_NUMBER_OF_T
HREADS][MAX_SYMBOLS].
```

- `void mainFunctionBuildTheTree()`

Тук се реализира основната логика на алгоритъма за компресия, т.е.:

1. Образуваме от всеки символ тривиално дърво, в корена (единствения връх) на което записваме честотата (вероятността) на срещане на съответния символ.
2. Намираме двата върха с най-малки честоти (вероятности) и ги обединяваме в ново дърво с корен, съдържащ сумата от честототите (вероятностите) им.
3. Ако има поне две дървета, преход към 2.

Най-важното свойство на дървото на Хъфман обаче е, че винаги гарантира построяването на оптимален побуквен код, затова и като структура от данни е избрана приоритетна опашка.

* Този алгоритъм не строй единствено дърво на Хъфман, защото може на една стъпка да имаме повече от един избор, а ние избираме коя да е от тях.

- `void constructTheMappingOfASCIIcodeToHuffmanTable(Node * root, std::string str)`

Това е рекурсивна функция, която прави обхождане на вече конструираното дърво на Хъфман, като записва в друга структура от данни (map) 1 към 1 съпоставянето на символа с неговия генериран код (т.е. пътя в дървото поредица от 0 и 1 в зависимост дали сме задълбали наляво или надясно).

- `void printTheCodes(), void decodedString() и void encodedString()`

Съветно принтират символите участващи във входящия текстов файл заедно с кода им, декодирания текстов файл или кодирания текстов файл.

4. Стартиране на програмата

Програмата се стартира от командния ред чрез подаване на следните параметри, като те самите могат да бъдат подадени в произволен ред:

- **-f <file_name: string>** (obligatory argument);
- **-t <thread_number: unsigned>** (nonobligatory argument) default value is 1;
- **-q** *(nonobligatory argument)
  default value is false;

- **-g <granularity: unsigned>** (nonobligatory argument) default value is 1;
- **-b <balancing: string>** (nonobligatory argument) default value is static;

* Когато е подаден параметърът -q, програмата се изпълнява в тих режим, т.е. извежда се само информация относно колко е цялото време за изчисление на честотната таблица. Когато не е подаден параметърът -q, програмата се изпълнява в нормален режим, като се извежда информация относно започването и приключването на всяка една нишка, времето й за изпълнение, размера на файла, от който ще се чете, размера на частта от файла, която всяка една нишка ще обработва, грануларността и декомпозицията на подзадания, режима на планиране и броят нишки, които ще бъдат използвани при това стартиране на програмата.

5. Резултати

Тестовете са правени върху текстов файл ~ 2GB.

По абсцисата на следващите графики са разположени броя нишки, а по ординатата – S_p , T_p или E_p.

Легенда:

T_1 - времето за изпълнение на серийната програма (или програмата използваща една нишка/един компютър).

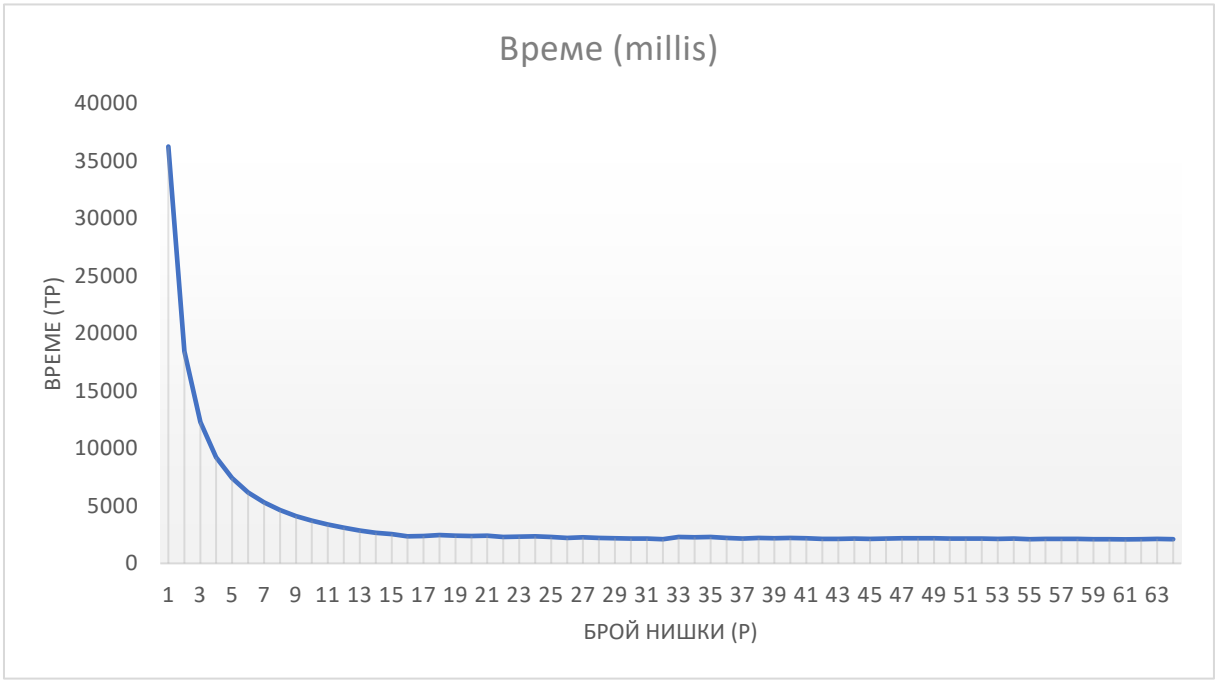
T_p - времето за изпълнение на паралелната програма, използваща p процесора/нишки/компютри.

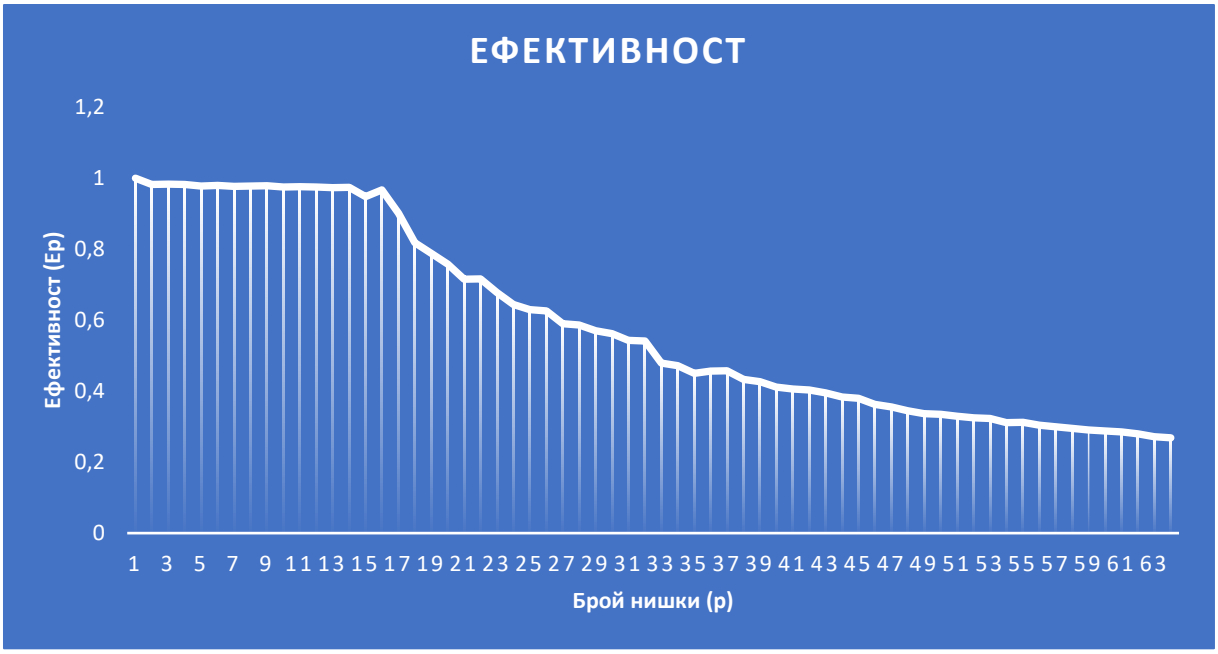
S_p- ускорението, което нашата програма има при използването на p процесора/нишки/компютри.

E_p - ефективността (ефикасността) на нашата програма, при използването на p процесора/нишки/компютри.

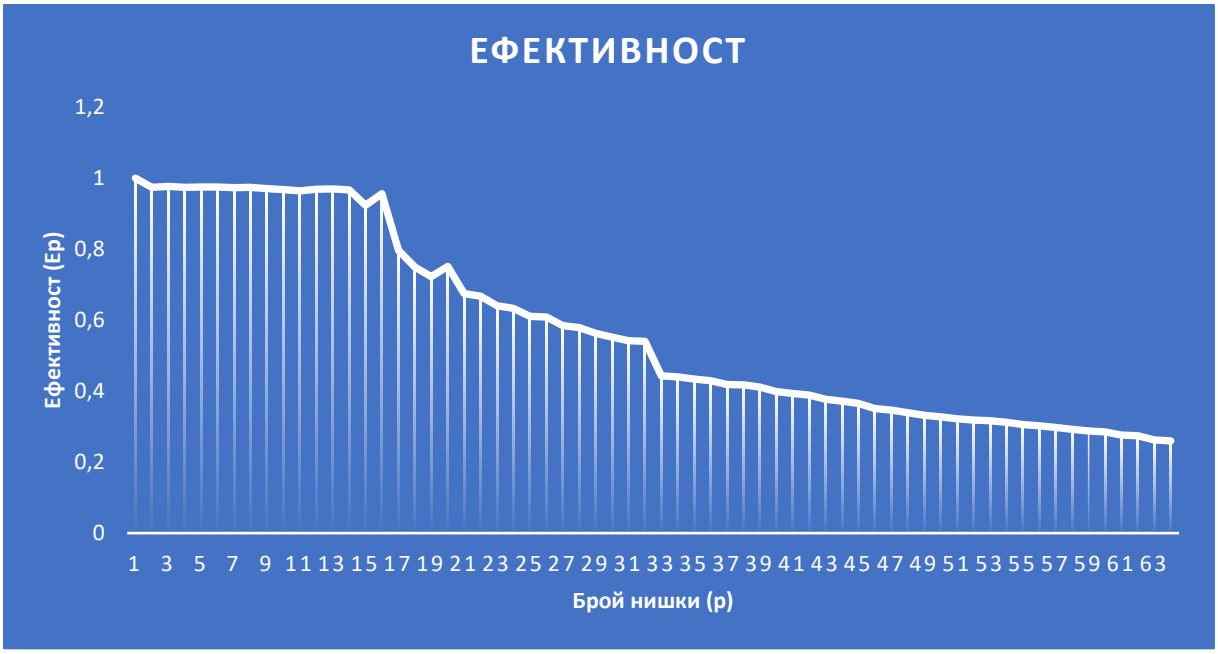
$S_p = T_1/T_p$, $E_p = S_p/p$, където p е паралелизма на програмата (процесора/нишки/компютри).

- Тествана със статично балансиране и грануларност 1(най-едра).

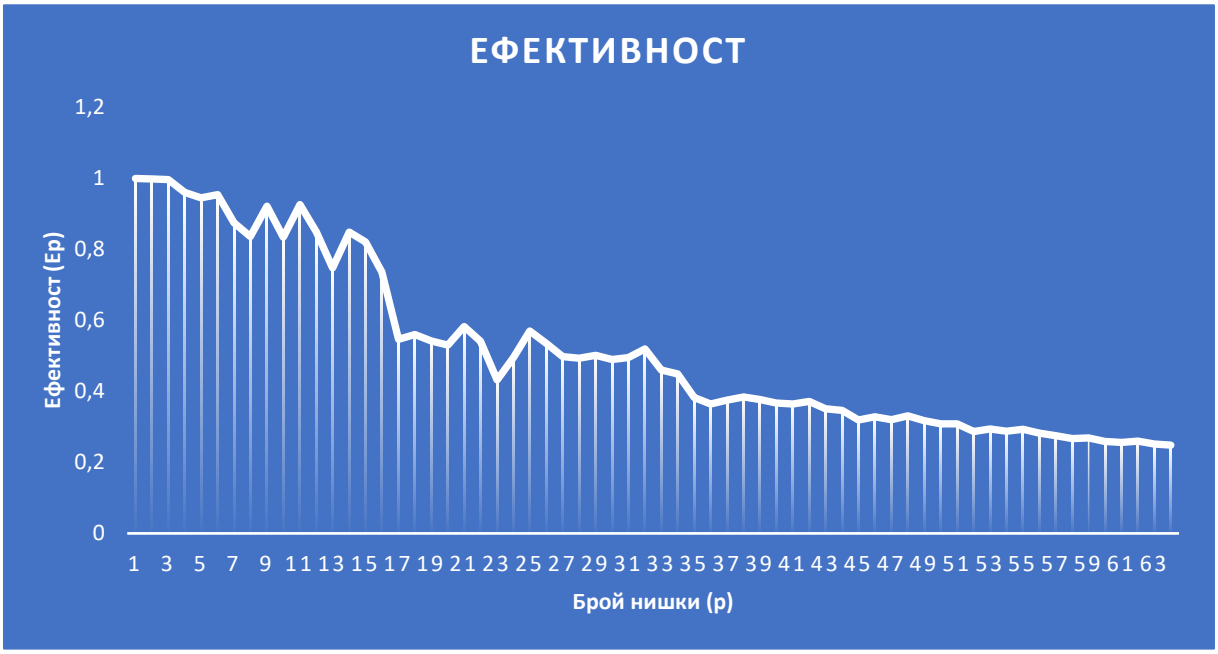



ЕФЕКТИВНОСТ
1,2
1
0,8
0,6
0,4
0,2
0
Ефективност (Ep)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63
Брой нишки (p)

- Тествана със статично балансиране с циклично планиране грануларност 3 (по-фина).


ЕФЕКТИВНОСТ
1,2
1
0,8
0,6
0,4
0,2
0
Ефективност (Ep)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63
Брой нишки (p)

- Тествана със динамично балансиране и грануларност 3 (по-фина).


ЕФЕКТИВНОСТ
1,2
1
0,8
0,6
0,4
0,2
0
Ефективност (Ep)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63
Брой нишки (p)