УДК 004.4:519.85

ГРАФІЧНА ПРОГРАМА-ПАРСЕР ДЛЯ АРИФМЕТИЧНИХ ВИРАЗІВ

Гацелюк Сергій

Науковий керівник: доцент, кандидат педагогічних наук Лупан І.В.

Центральноукраїнський державний педагогічний університет імені Володимира Винниченка, м. Кропивницький, Україна

В статті описано програму-парсер арифметичних виразів, яка є результатом виконаного курсового проекту з комп'ютерних наук. Програма виконує аналіз та візуальзацію формули, представленої виразом, з метою визначення можливостей її розпаралелювання та обчислює характеристики (прискорення та ефективність) відповідного паралельного алгоритма. Програма може бути використана для унаочнення процесу розпаралелювання при вивченні відповідних навчальних курсів. У статті наведено лістинги деяких фрагментів програми та результати застосування програми до деяких виразів.

Ключові слова: *арифметичний вираз, парсер, паралельний алгоритм, прискорення паралельного алгоритма, ефективність паралельного алгоритма.*

THE PROGRAM FOR A GRAPHICAL PARSER OF ARITHMETIC EXPRESSIONS

Gatseliuk Sergiy

Scientific supervisor: Candidate of Pedagogical Sciences, Docent Lupan I.V.

Volodymyr Vynnychenko Central Ukrainian State Pedagogical University, Kropyvnytsky, Ukraine

The article describes a program-parser of arithmetic expressions, which is the result of a completed course project in computer science. The program analyzes and visualizes the formula represented by the expression in order to determine the possibilities of its parallelization and calculates the characteristics (acceleration and efficiency) of the corresponding parallel algorithm. The program can be used to visualize the parallelization process when studying the relevant training courses. The article lists some parts of the program and the results of applying the program to some expressions.

Keywords: *arithmetic expression, parser, parallel algorithm, acceleration of parallel algorithm, efficiency of parallel algorithm*

Постановка проблеми. Парсинг (англ. parsing) – синтаксичний аналіз – це процес аналізу вхідної послідовності символів з метою розбору граматичної структури та оформлення її у структуру даних, зазвичай – в дерево, яке відповідає синтаксичній структурі вхідної послідовності і добре підходить для подальшої обробки. Синтаксичні аналізатори є основною складовою компіляторів та інтерпретаторів, тобто програм для розпізнавання вхідної інформації [1, 4].

У даній роботі парсинг здійснюється з метою виявлення можливостей для розпаралелювання процесу обчислення арифметичних виразів та визначення характеристик паралельного алгоритма. Задача є актуальною при вивченні дисципліни «Паралельні та розподілені обчислення».

Мета дослідження полягала у створенні комп'ютерної програми, яка дозволить здійснювати парсинг арифметичного виразу, візуалізувати дерево виразу та визначати характеристики паралельного алгоритма (тобто необхідну кількість процесорів, прискорення, ефективність), який може бути застосований до обчислення даного виразу при наявності достатньої кількості процесорів.

Постановка задачі. *Нехай E – простий арифметичний вираз, що задовольняє умові:*"*Кожна змінна входить в E рівно один раз*". (*)

Необхідно побудувати та візуалізувати дерево формули для даного виразу та розрахувати для нього необхідну кількість процесорів, кількість ярусів дерева (кількість паралельних кроків), а також визначити прискорення та ефективність паралельного алгоритму.

Найбільш відомі алгоритми розпаралелювання арифметичних виразів Баєра-Бовета, Брента і Винограда [5] засновані на загальному принципі: *орієнтований ациклічний граф, що описує послідовне обчислення виразу E, представляє собою, з урахуванням властивості* (*), *бінарне дерево*.

Отже, *основна мета* розпаралелювання арифметичних виразів – розробити дерево обчислень мінімальної висоти. Тобто, за умови реалізації на кожному ярусі максимальної кількості незалежних операцій, вираз має бути обчислений за мінімально можливий час або за мінімальну кількість кроків. Розпаралелювання АВ здійснюється в рамках моделі MIMD (Multiple Instruction Multiple Date – множинний потік команд і множинний потік даних) з необмеженим паралелізмом.

Порядок обчислення може бути перерозподілений або довизначений таким чином, щоб деякі операції можна було виконувати паралельно. Тому говорять, що вирази мають *неявний паралелізм*.

Характеристиками складності обчислення арифметичного виразу є:

– час t, що витрачається на обчислення арифметичного виразу;
– загальна кількість операцій w, необхідна для обчислення виразу;
– кількість процесорів (p), необхідна для реалізації обчислень.

Якщо вважати, що будь-яка операція виконується за одну одиницю часу, то час ***t*** обчислення арифметичного виразу дорівнює числу ярусів (висоті) дерева обчислень. Операції, що знаходяться на одному ярусі дерева, можуть виконуватися паралельно та визначають ширину даного яруса. Тому висота дерева відповідає числу кроків паралельного алгоритму для арифметичного виразу.

Потрібна кількість обчислювачів (або процесорів) ***p*** визначається як максимальна ширина яруса дерева обчислень.

Для оцінки розпаралеленого виразу використовують такі характеристики паралельності, як ***прискорення*** та ***ефективність*** паралельних алгоритмів.

Нехай n – число параметрів задачі (для арифметичного виразу – число різних змінних, що входять в арифметичний вираз).

$T_p(n)$ – час виконання паралельного алгоритму на обчислювальній системі з числом процесорів p>1.

$T_1(n)$ – час виконання "найкращого" послідовного алгоритму.

Прискоренням S_p паралельного алгоритму називають відношення $S_p(n) = \frac{T_1(n)}{T_p(n)}$, а ***ефективність*** E_p паралельного алгоритму визначається формулою $E_p(n) = \frac{S_p(n)}{p}$ [2, 3].

Технічне завдання полягало у тому, щоб створити програму, яка за введеним арифметичним виразом, що містить операнди (у вигляді літер латинського алфавіту, причому розглядаються лише однобуквенні операнди), знаки арифметичних операцій (+, –, *, /) та круглі дужки, будуватиме дерево обчислень та за параметрами побудованого дерева визначатиме характеристики паралельного алгоритма.

При розробці алгоритма довелося стикнутися з такими проблемами:

1) врахування пріоритетів арифметичних операцій у виразах без дужок та зміни приіоритетів у виразах з дужками;
2) урахування порядку виконання ланцюжків операцій однакового пріоритету
3) з'єднанні на рисунку операторів різних ярусів gри розробці алгоритма візуалізації дерева.

Ці проблеми було вирішено в рамках виконаної курсової роботи. В результаті програма добре справляється з виразами, які дозволяють використання схеми здвоєння. Довжина таких виразів в принципі не обмежується розміром текстового вікна (25 операндів). Однак для виразів з операціями різного пріоритету та дужками є певні обмеження.

Наведемо декілька виразів, з якими програма працює коректно, та результати їхньої обробки та візуалізації програмою-парсером (Рис. 1, Рис. 2, Рис. 3):

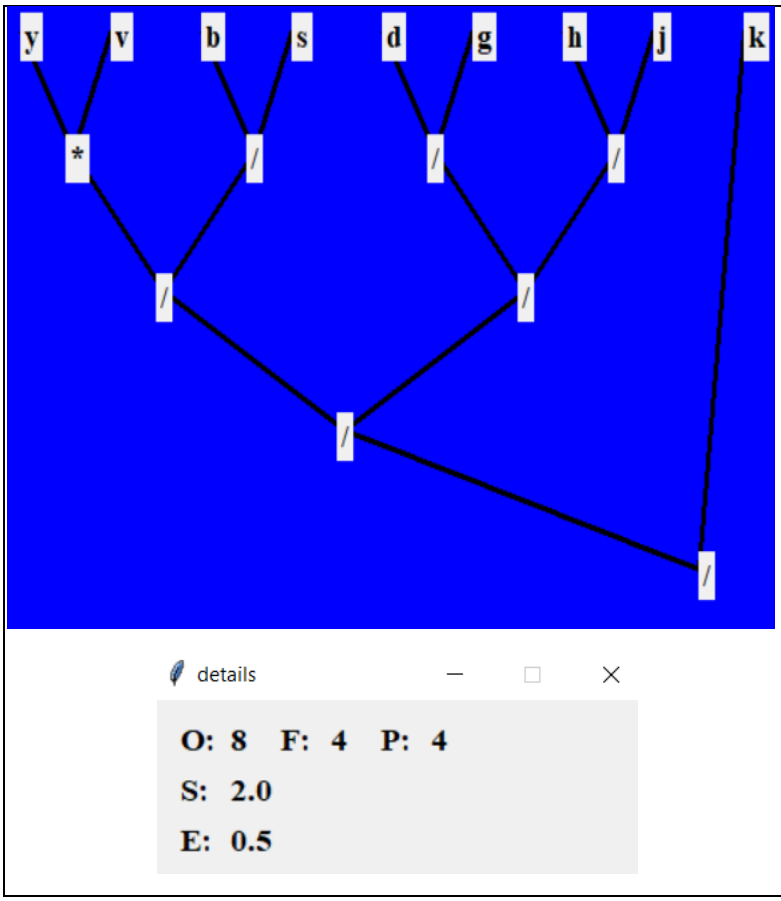

Рис. 1. Вираз y*v/(b/s)/d/g/h/j/k

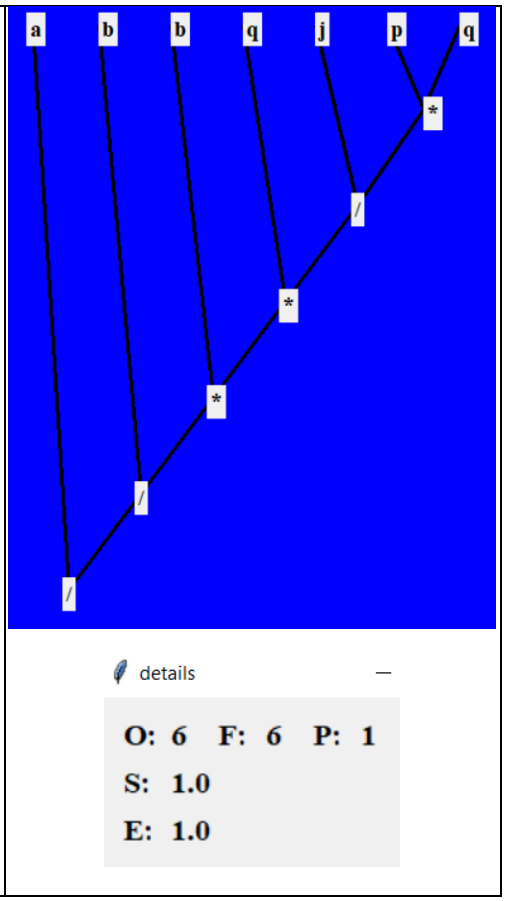

Рис. 2. Вираз a/(b/(b*(q*(j/(p*q)))))

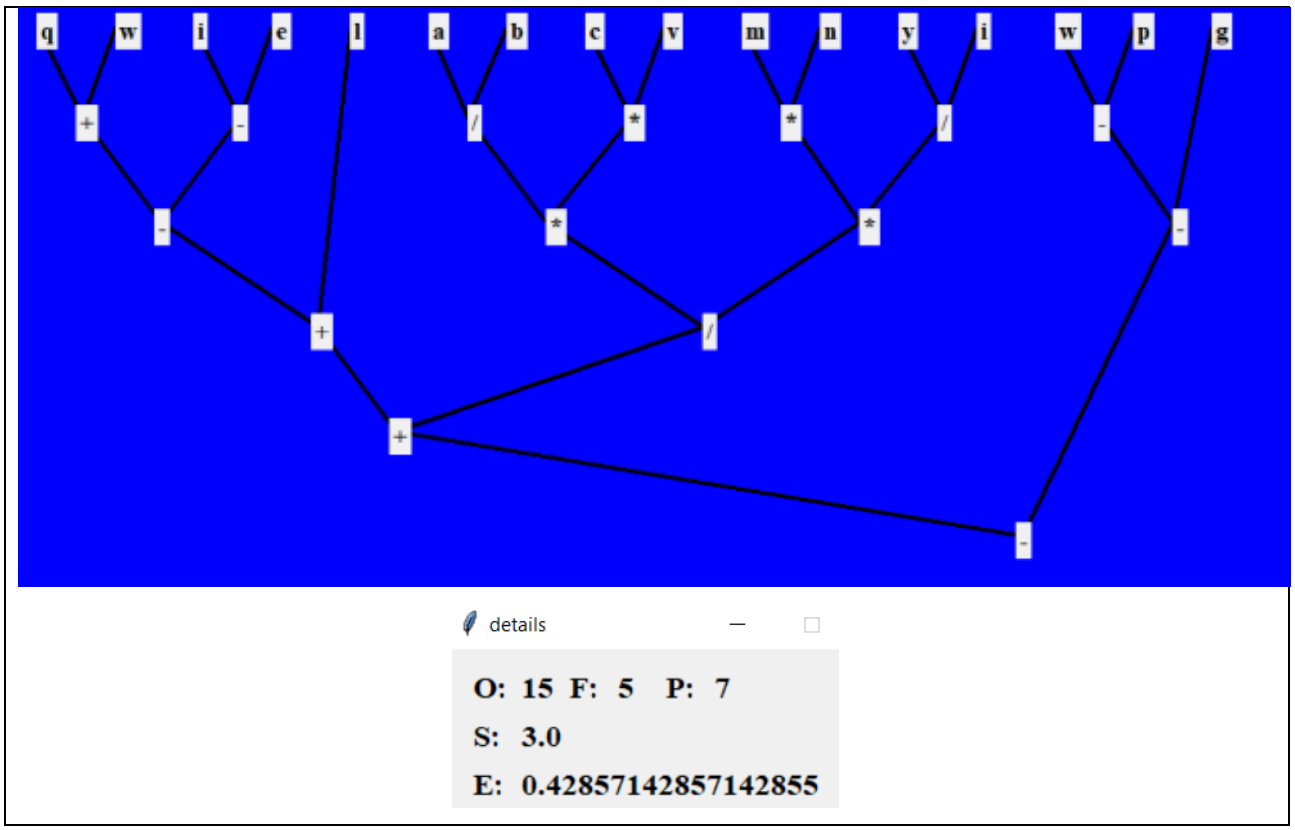

Рис. 3. Вираз q+w-i-e+l+(a/b*c*v/m*n*y/i)-w-p-g

При роботі з програмою слід дотримуватися таких правил:

Правило 1 (використання літер): Програма розпізнає змінними лише літери латинського алфавіту, як нижнього так і верхнього регістру. Літери інших розкладок не враховуються і програма не буде їх виводити на екран при побудові дерева. Однак при бажанні, можна додати більше літер в масив alphabetka, і тоді програма буде приймати ті змінні, які внесли.

Правило 2 (використання операндів): Програма розпізнає операндами лише чотири символи:

“*” – множення (код символа 002A);

“/” – ділення (код символа 002F);

“-“ – віднімання (код символа 002D);

“+” – додавання (код символа 002B).

Спеціальні символи, які мають схоже зображення, або семантичний смисл (наприклад, “×” як знак множення), не сприймаються програмою, тому не будуть виведені у вікні Tree та враховані у парсингу.

При бажанні, можна додати й інші варіанти операндів, однак для цього доведеться додавати їх в масиви operands та operands_with_hooks, де вони зустрічаються, для всіх циклів, що пов'язані з операндами доведеться змінювати обмеження на кількість операндів, а в умовах, де перевіряється символ, потрібно буде додати нові операнди, щоб вони також враховувались.

Правило 3 (використання дужок): Слід розставляти дужки згідно з правилами математики: кількість відкритих дужок має співпадати з кількістю закритих. Програма може прочитати навіть неправильно введений вираз, однак результат буде неправильним. І ніколи не слід ставити ")" перед "(", бо програма може не вивести операнди.

Правило 4 (порядок введення виразів): Введення зазвичай починається з назви змінної, або з "(" і потім змінної. Далі вводиться один з чотирьох операндів, після чого зазвичай йде змінна, однак це також може бути "(", якщо вираз продовжується, або ")" якщо закривається локальний вираз, або закривається весь вираз, і в другому варіанті може бути кількість ")" рівна відкритим "(" попередньо, однак вони не повинні бути вже закритими. В кінці виразу, кількість "(" та ")" повинна бути однаковою, а кількість операндів має бути на одиницю менше кількості змінних.

Програму було розроблено мовою Python [6], а інтерфейс створено з використанням вбудованого в Python модуля tkinter.

Логіку побудови дерева формули реалізовано у функції Tree, лістинг якої наведено нижче (Лістинг 1):

Лістинг 1:

```
#$---------------------------Tree-----------------------------------------------$
def Tree():
    tk2=Toplevel(tk0)
    tk2.geometry('800x700')
    frame=Frame(tk2, width=10000, height=10000)
    frame.pack(fill=BOTH)
    canvas=Canvas(tk2,bg='blue',width=10000,height=10000,scrollregion=(0,0,10000,10000))
    tk2.title("Tree")

    scrollTL=Toplevel(tk2)

    hbar=Scrollbar(frame,orient=HORIZONTAL)
    hbar.pack(side=BOTTOM,fill=X)
    hbar.config(command=canvas.xview)
    vbar=Scrollbar(canvas,orient=VERTICAL)
    vbar.pack(side=RIGHT,fill=Y)
    vbar.config(command=canvas.yview)
    canvas.config(width=300,height=300)
    canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)
    canvas.pack(side=LEFT, expand=True,fill=BOTH)

    equency = edit0.get()
```

Щоб побудувати дерево формули, потрібно було створити функцію-парсер для змінних та операндів, які вводить користувач в форму Entry на головному екрані. Серед різних способів парсинга, ми обрали парсинг з пріоритетами. Ця функція буде складатись з декількох частин: вибірки змінних з рядка, вибірки операндів з рядка, вибірки операндів та дужок, надання пріоритетності операндам, виведення змінних на екран, поєднання пріоритетів і операндів в один масив, лічильника однакових пріоритетів операцій – та +, лічильника однакових пріоритетів операцій / та *, функції побудови дерева для одного операнда, функції побудови дерева для 1+ операндів (з 2 частин, де 1-ша для – та +, друга – для / та *) (Лістинг 2):

Лістинг 2:

```
    alphabetka = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s
    operands = ['-','+','*','/']
    operands_with_hooks = ['(',')','-','+','*','/']
#-------------------------------------------------
    Library = []#Out letters here
#-------------------------------------------------
    OperLibrary = []#Out operands here
    HOperLibrary = []# Out operands with hooks
    Priority_Of_Operands = []
    OperandsXCoord = []# There are coords X
    Floor = []
```

Перш за все після введення арифметичного виразу слід розсортувати всі операнди за пріоритетністю та занести їхні параметри у масив MainMass.

Далі були виконані обрахунки для визначення ярусності, кількості процесорів та операндів. Після цього за формулами, наведеними вище, були обчислені ефективності та прискорення. Детально алгоритм роботи програми наведено у курсовій роботі.

Висновки. Відповідно до поставлених задач у результаті виконання курсової роботи було створено програму-парсер арифметичних виразів, яку можна використовувати для демонстрації можливостей розпаралелювання обчислення арифметичних виразів. Оптимізація розпаралелювання при написанні даної програми не враховувалася. Дерево формули будується за вхідним виразом. На жаль тестування деяких виразів показало, що залишилися випадки, в яких програма працює не зовсім коректно. У подальшій роботі планується здійснити більш глибокий аналіз граматики арифметичних виразів з

метою побудови більш досконалого алгоритма, можливо із залученням скінченого автомата для синтаксичного аналізу виразів. Також планується створити програму, яка буде оптимізувати вхідний вираз з метою покращення характеристик паралельного алгоритма, тобто зводити подібні доданки, застосовувати асоціативні закони виконання арифметичних операцій, розкривати дужки тощо.

Список використаної літератури

1. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципи, технологии и инструментарий = Compilers: Principles, Techniques, and Tools. – 2-е изд. – М.: Вильямс, 2008.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – С-Пб: "БХВ-Петербург", 2002. – 608 с.
3. Кузьменко Б.В., Чайковська О.А. Технологія розподілених систем та паралельних обчислень. (Конспект лекцій, частина 1. Розподілені об’єктні системи, паралельні обчислювальні системи та паралельні обчислення, паралельне програмування на основі MPI) Навчальний посібник. – К.: Видавничий центр КНУКІМ, 2011. – 126 с.
4. Робин Хантер. Основные концепции компиляторов = The Essence of Compilers. – М.: "Вильямс", 2002. - С. 256.
5. Spiewak Daniel. Generalized Parser Combinators. – 2010. – URL: https://github.com/djspiewak/ gll-combinators.
6. http://python.org