Факултет техничких наука

Универзитет у Новом Саду

Рачунарски системи високих перформанси

Паралелизација проблема проналажења суме у бинарном стаблу

Аутор:
Бојан Попржен

Индекс:
Е2 4/2022

16. јануар 2023.

Сажетак

У овом раду анализиран је проблем проналажења суме у бинарном стаблу. Проблем проналажења суме у бинарном стаблу можемо представити питањем: "Да ли чворови бинарног стабла којима је придружен природан број у било којој путањи од корена до листа дају задати збир?". Поступак за решавање овог проблема могуће је разложити на независне делове које је могуће паралелно решити. Решење проблема приказано у овом раду пружа приближно линеарно убрзање доласка до решења са порастом броја процесуирајућих јединица.

Садржај

Списак изворних кодова

Списак слика

Списак табела

1 Увод

У овом раду, анализиран је проблем проналажења суме у бинарном стаблу. Проблем можемо представити и питањем: "Да ли чворови бинарног стабла којима је придружен природан број у било којој путањи од корена до листа дају задати збир?".

Најчешће решење овог problema има асимптотску временску сложеност извршавања $O(n)$ gde n представља величину улаза tj. број чворова у графу.

У овом раду, предложено је паралелно решење проблема које узима у обзир независност обраде деце одређеног чвора.

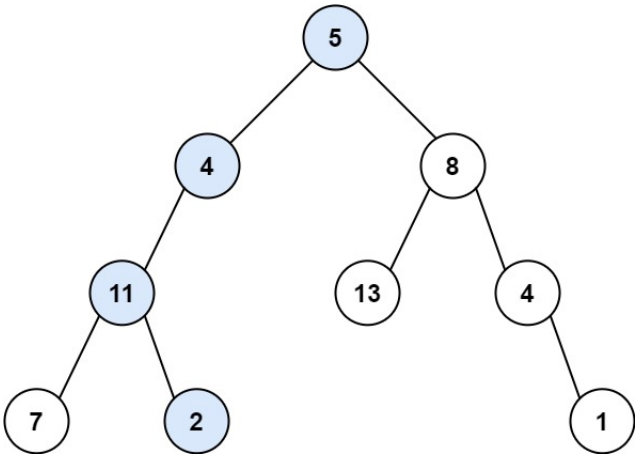
Постигнута асимптотска сложеност паралелног решења је приближно: $O(n/p)$, где је n величина улаза тј. број чворова у графу, а p број јединица паралелног извршавања.

Рад је конципиран на следећи начин: прво ће сам проблем бити описан као и његово секвенцијално решење, потом ће бити приказано паралелно решење и , на крају, биће представљени подаци о ефикасности секвенцијалног и паралелног решења.

2 Проблем проналаска суме у бинарном стаблу

2.1 Опис проблема

Стабло је граф који је повезан и који нема циклуса. Бинарно стабло је усмерено стабло у коме сваки чвор, почевши од коренског чвора, има 0, 1 или 2 деце-чворова. Дететом чвора називамо чвор на кога неки други чвор показује. Родитељем чвора називамо чвор који на неки други чвор показује. Листом називамо чвор који има 0 деце.



Слика 1: Пример постојања тражене путање за проблем проналаска суме у бинарном стаблу

У примеру 1, плавом је приказана једна путања која даје тражену суму **22**.

	Улаз
1	Бинарно стабло
2	Тражена сума

Табела 1: Табела улаза за проблем проналаска суме у бинарном стаблу

	Излаз
1	Булова променљива

Табела 2: Табела излаза за проблем проналаска суме у бинарном стаблу

За проблем су дата два улаза (табела 1) - стабло и тражена сума, док је тражен један излаз (табела 2) - Булова променљива чија је тачност еквивалентна чињеници

да је решење пронађено, односно да за дати граф постоји путања од корена до листа која даје тражену суму.

2.2 Секвенцијално решење проблема

Најчешће секвенцијално решење проблема своди се на обилазак графа по дубини (енг. *depth first search*, DFS).

Секвенцијално решење прво обради (по дубини) лево дете-чвор па потом десно. Обрада појединачног чвора своди се на следећи поступак:

Постави тренутан чвор на УЛАЗ 1 (корен стабла). Постави тренутно тражену суму на УЛАЗ 2 (тражену суму). Покрени обраду описану корацима:

1. Уколико је тренутан чвор нула-чвор, врати НЕТАЧНО.
2. Уколико тренутан чвор није нула-чвор:
   (a) Уколико је број придружен чвору већи од тренутно траженог збира, врати НЕТАЧНО.
   (b) Уколико је број придружен чвору једнак тренутно траженом збиру и тренутни чвор је лист, врати ТАЧНО.
   (c) Уколико је број придружен чвору мањи или једнак тренутно траженом збиру, умањи тренутно тражени збир за тај број. А потом:
      i. Уколико обрада левог детета-чвора за тренутно тражену суму врати ТАЧНО, врати ТАЧНО.
      ii. Уколико обрада десног детета-чвора за тренутно тражену суму врати ТАЧНО, врати ТАЧНО.

У идеалном случају, решење представља путања састављена од свих левих детета-стабала. У том случају до решења се долази у асимптотском времену $(logn)$, где је n број чворова у стаблу. Још једна од карактеристика секвенцијалног решења јесте да се решење не тражи у десном детету-стаблу уколико је оно пронађено у левом детету-стаблу. У најгорем (општем) случају ово решење има **асимптотску временску сложеност извршавања** $O(n)$ gde n представља величину улаза tj. број чворова у графу.

Програмски код за секвенцијално решење дат је у листингу 1. Напомена: изостављен је део кода за учитавање библиотека, учитавање и креирање графа.

```c
bool traverse(struct TreeNode* root, u_int64_t* sum) {
    if(root==NULL) return false;

    (*sum) -= root->val;
    if(root->right==NULL&&root->left==NULL&&*sum==0)
    {
        return true;
    }
    u_int64_t lsum, rsum;
    lsum = rsum = *sum;
    return traverse(root->left, &lsum) || traverse(root->right, &rsum);
}
```

Изворни код 1: Имплементација секвенцијалног решења у језику C

3 Паралелно решење проналаска суме у бинарном стаблу

3.1 Опис паралелног решења

Дефинишимо *дете-стабло* као стабло чији је корен дете датог чвора.

Секвенцијално решење не користи чињеницу да се "лево" и "десно" дете-стабло могу независно обрађивати. Тачније, можемо измрестити (енг. *spawn*) две независне нити извршавања где прва обрађује лево, а друга десно дете-стабло. Спајање (енг. *join*) нити

Међутим, у имплементационом смислу, оптимално решење није да се свако дете-стабло обрађује независно тј. у посебном ОпенМП задатку, јер тиме се оптерећује извршно окружење ОпенМП-а тј. доста процесорског времена се потроши на прекључивање контекста задатака уместо на ефективном раду и извршавању задатака.

Предлаже се паралелно решење које независно обрађује стабла чији су корени на нивоу l, таквом да:

$$\begin{aligned} & l = max_i 2^i \leq \text{omp\_get\_num\_threads}() \iff \\ & l = max_i i \leq log_2 \text{omp\_get\_num\_threads}() \iff \\ & l = \lfloor log_2 \text{omp\_get\_num\_threads}() \rfloor. \end{aligned} \tag{1}$$

Интуиција иза овога јесте да **паралелно** можемо да обрађујемо само онолико стабала колико имамо процесорских јединица на располагању. Стога, треба да паралелно обрађујемо стабла са коренима на нивоу ком је број чворова такав да већ следећи ниво стабла има више чворова него што програм има процесорске моћи.

3.2 Аспекти OpenMP библиотеке

За имплементацију паралелног решења, коришћена је OpenMP библиотека. Главна директива OpenMP библиотеке која је коришћена јесте задатак (енг. *Task*). Задатак је блок кода које извршно окружење OpenMP-а распоређује и извршава.

Дужност програмера је да назначи који блок кода представља задатак директивом `#pragma omp task`. Када извршавање изворног кода "дође" до ове директиве, извршавање наредног блока преузима OpenMP, гарантујући да ће се он извршити до следеће имплицитне или експлицитне директиве за завршетак извршавања задатака. Ова директива може да се експлицитно наведе као: `#pragma omp taskwait`.

Начелно, не постоји ограничење на број задатака који могу да се извршавају конкурентно у OpenMP извршном окружењу. Међутим, паралелно може да се извршава онолико задатака колико постоји процесуирајућих јединица на рачунару на ком се програм извршава.

У самој имплементацији паралелног решења, како би се паралелно обрађивала стабла на нивоу l, **морамо да креирамо задатке и за обраду чворова на нивоима мањим од** l. То је једини начин да се независно креирају задаци за све чворове нивоа l. У супротном случају, обрада би текла тако да се независно обради први пар чворова нивоа l, потом други итд. То значи да је укупан број задатака за обраду једнак $2^(l + 1) - 1$, односно, у случају да рачунар на ком се покреће решење има 4 процесуирајуће јединице, креираће се 7 задатка, на рачунару са 8 ПЈ креираће се 15 задатака итд.

3.3 Паралелно решење

У листингу 2 представљено је паралелно решење.

Асимптотска сложеност овог решења је приближно:

$$O(n/p),$$

где је n величина улаза тј. број чворова у графу, а p број јединица паралелног извршавања.

```c
#define MAX_THREADS 12

bool traverse_parallel(struct TreeNode* root, u_int64_t* sum, int l, int curr_l)
{
    if(root==NULL) return false;

    (*sum) -= root->val;
    if(root->right==NULL&&root->left==NULL&&*sum==0)
    {
        return true;
    }
    u_int64_t lsum, rsum;
    lsum = rsum = *sum;
    bool left, right;

    if (curr_l <= l) {
        #pragma omp task shared(left)
        {
            left = traverse_parallel(root->left, &lsum, l, curr_l+1);
        }
        #pragma omp task shared(right)
        {
            right = traverse_parallel(root->right, &rsum, l, curr_l+1);
        }
        #pragma omp taskwait
    } else {
        left = traverse(root->left, &lsum);
        right = traverse(root->right, &rsum);
    }
    return left || right;
}

bool pathSum(struct TreeNode* root, u_int64_t* sum){
    bool res;
    #pragma omp parallel num_threads(MAX_THREADS) shared(res)
    {
        #pragma omp single
        {
            int l = floor(log(omp_get_num_threads()) / log(2)) - 1;
            res = traverse_parallel(root, sum, l, 0);
        }
    }
    return res;
}
```

Изворни код 2: Имплементација паралелног решења у језику C

4 Експеримент

У овом поглављу описан је експеримент који треба да измери перформансе секвенцијалног, односно паралелног решења. Такође, представљен је преглед резултата експеримента, то јест, упоредни приказ оба решења.

4.1 Опис експеримента

Експеримент описан у овом поглављу треба да измери перформансе секвенцијалног и паралелног решења у различитим случајевима коришћења.

Проблем проналаска суме у бинарном стаблу генерално је веома незахтеван што се тиче процесорских ресурса. То се дешава због чињенице да је обрада појединачног чвора веома краткотрајна у погледу процесорског времена. Стога, како би проблем било доследније измерити [1], у секвенцијално и паралелно решење додато је време обраде појединачног чвора од $1\mu s$.

Природа паралелног решења је таква да су сви OpenMP задаци

Паралелно решење покретано је са **8, 4 и 2 процесуирајуће јединице**. Бројева 8, 4 и 2 су изабрани због тога што су то степени броја два тј. бројеви чворова на нивоима 3, 2 и 1 бинарног стабла, респективно, те ова решења нуде могућност паралелне обраде l чворова. Међутим, решење је покретано и са 12 ПЈ, јер се при иницијалним мерењима показала предност покретања паралелног решења и са бројем ПЈ који није умножак двојке, а која се своди на следеће: сви OpenMP задаци у паралелном решењу познати су тек када сви задаци на нивоу $l-1$ започну своје извршавање и генеришу те задатке; OpenMP извршно окружење фаворизује редно извршавање задатака, онај задатак који се генерише раније, он ће се раније и извршити. Последица свега овога јесте да

Решења су покретана над стаблима величина 100 хиљада чворова, 10 хиљада чворова и 9 чворова (пример са слике 1). Тражена сума је бирана тако да се:

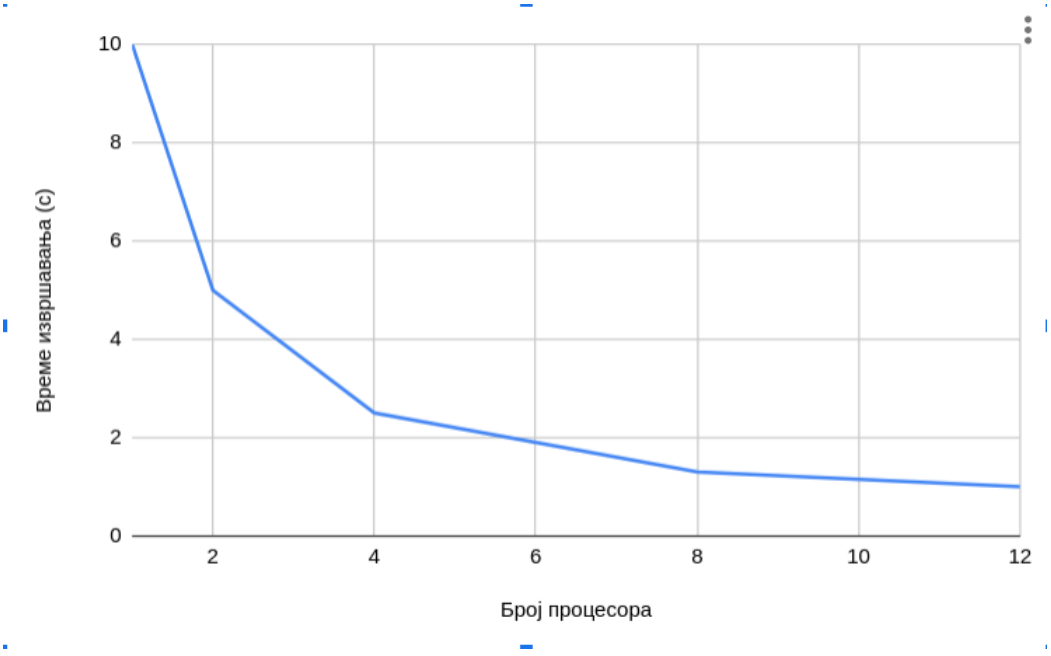
1. Ne може доћи до решења.

2. Сума налази на путањи састављеној од све леве деце-чворова (оптималан случај секвенцијалног решења). Означено као **решење Л**.

3. Сума налази на путањи састављеној од деце-чворова биране тако да се наизменично бирају лево и десно дете-чвор (погодан случај паралелног решења). Означено је као **решење С**.

[1] Приликом иницијалних мерења, оба решења су изузетно брзо долазила до краја извршавања. Дешавало се да времена извршавања једног решења буду многоструко различита. То се дешавало због стохастичних процеса попут процесорског кеширања, избора задатка за извршавање од стране OpenMP библиотеке итд. који су од извршавања до извршавања били мање или више повољнији.

4.2 Резултати



Слика 2: Резултати покретања решења са растућим бројем процесуирајућих јединица

На слици 2 приказана је брзина извршавања решења у односу на растући број процесуирајућих јединица. Секвенцијално решење представљено је подацима о извршавању са 1 процесуирајућом јединицом. Перформансе извршавања секвенцијалног решења су, наравно, независне од броја процесуирајућих јединица.

5 Закључак

У овом раду, анализиран је проблем проналажења суме у бинарном стаблу. Дат је приказ проблема и његовог секвенцијалног решења. Потом је приказано и његово паралелно решење. На крају, приказани су резултати који указују на линеарно убрзање доласка до решења са порастом броја процесуирајућих јединица.