

Laboratorinis darbai

Domantas Keturakis

Spalis 2024

UŽDUOTIS #1

Pagrindinės įžvalgos

Pirmasis skaičiavimo skaičiavimo ciklas (Pav. 1) palyginus užtrunka nedaug laiko (apie 0.002 sekundės). Praktiniams tikslams, jį galima ignoruoti.

```
for (int i=0; i<numX; i++) {  
    X[i] = i;  
    bestX[i] = i;  
}  
u = evaluateSolution(X);  
bestU = u;
```

Pav. 1: Pradinės naujo ir geriausio sprendinių reikšmių apskaičiavimas

Didžiąją dalį laiko užima šis ciklas:

```
while (increaseX(X, numX-1, numCL) == true) {  
    u = evaluateSolution(X);  
    if (u > bestU) {  
        bestU = u;  
        for (int i=0; i<numX; i++) bestX[i] = X[i];  
    }  
}
```

Pav. 2: Visų galimų sprendinių perrinkimas

Šį ciklą iš esmės yra ne paprasta parelilizuoti, nes:

- `increaseX` - keičia masyvo `X` reikšmę (Pav. 3), ir ne tik `index`-ąjį elementą, rekursyviai kviesdamas save sumažina `index` reikšmę vienu, t.y. iškvietus `increaseX` visos masyvos reikšmės yra keičiamos. To pasekmė, kad `index`-ojo elemento skaičiavimo negalima paskirstyti skirtingoms gijoms, kitaip vėlesnėms gijoms reikėtų laukti, kol praeita gija baigs savo darbą, visiškai nustelbiant paralelizavimo naudą.
- `increaseX` skaičiavimai priklauso vienas nuo kito, t.y. norint apskaičiuoti `X` reikšmę n -ame ciklo vykdyme, reikia pirma apskaičiuoti `X` reikšmę $(n - 1)$ -ame ciklo vykdyme. Analogiškai negalima paralelizuoti ir nes kitos gijos lauktų, kol praeita gija baigs savo darbą.
- `if (u > bestU) { ... }` irgi gali tik vienas ciklas vienu metu, nes `bestU` ir `X` pakeitimas turi būti atliekamas "žingsniu".

Iš esmės neperrašius `increaseX`, šios funkcijos ir jos kvietimo cikle, yra nepraktiška parelilizuoti.

```

int increaseX(int *X, int index, int maxindex) {
    if (X[index]+1 < maxindex-(numX-index-1)) {
        X[index]++;
    }
    else {
        if ((index == 0) && (X[index]+1 == maxindex-(numX-index-1))) {
            return 0;
        }
        else {
            if (increaseX(X, index-1, maxindex)) X[index] = X[index-1]+1;
            else return 0;
        }
    }
    return 1;
}

```

Pav. 3: funkcijos increseX apibrėžimas

Tuo tarpu funkcija evaluateSolution nekeičia jokių globalių kintamųjų ar savo argumentų. Anali-tiškai žiūrint galima spėti, kad čia ir didžioji dalis skaičiavimo laiko yra sugaištama. Teorinis šios funkcijos “big-O” yra $O(\text{numDP} \cdot \text{numX})$, tuo tarpu increseX rekursyviai save gali iškviesti daugiausiai numX kartų.

```

double evaluateSolution(int *X) {
    double U = 0;
    double totalU = 0;
    int bestPF;
    int bestX;
    double d;

    #pragma omp parallel reduction (+:U, totalU) private(bestPF, bestX, d)
    #pragma omp for
    for (int i=0; i<numDP; i++) {
        totalU += demandPoints[i][2];
        bestPF = 1e5;
        for (int j=0; j<numPF; j++) {
            d = HaversineDistance(i, j);
            if (d < bestPF) bestPF = d;
        }
        bestX = 1e5;
        for (int j=0; j<numX; j++) {
            d = HaversineDistance(i, X[j]);
            if (d < bestX) bestX = d;
        }

        if (bestX < bestPF) U += demandPoints[i][2];
        else if (bestX == bestPF) U += 0.3*demandPoints[i][2];
    }
    return U/totalU*100;
}

```

Pav. 4: Paraleлізуota funkcija evaluateSolution

Pav. 4 esantį for ciklą galima parelilizuoti, kadangi bestPF, bestX, d kintamiesiems ciklo viduje prieš jų panaudojimą priskiriamos tų pačių konstantų reikšmės ir kaip minėta X reikšmė nekeičiama. Kiekvienai gijai sukuriant atskirą bestPF, bestX, d kopiją išvengiamos “data-race” problemos. Už for ciklo ribų reikšmingi tik totalU ir U kintamieji, juos apsaugi nuo “data-race” galima apsaugti panaudojant reduce direktyvą, kadangi jos naudojamos tik galutiniui rezultatui susumuoti.

Rezultatai #1

NUM_THREADS = 1

Matricos skaiciavimo trukme: 4.66416
Sprendinio paieskos trukme: 17.2353
Algoritmo vykdymo trukme: 21.8994

Matricos skaiciavimo trukme: 4.70804
Sprendinio paieskos trukme: 17.0992
Algoritmo vykdymo trukme: 21.8072

Matricos skaiciavimo trukme: 4.70465
Sprendinio paieskos trukme: 17.4169
Algoritmo vykdymo trukme: 22.1216

Matricos skaiciavimo trukme: 4.70014
Sprendinio paieskos trukme: 17.5637
Algoritmo vykdymo trukme: 22.2638

Matricos skaiciavimo trukme: 4.6489
Sprendinio paieskos trukme: 17.3402
Algoritmo vykdymo trukme: 21.9891