
Laborator 3

Paralelizarea buclelor de tip for

Problemă de rezolvat

Să se rezolve un număr de 25.000.000 de ecuații de gradul 2.

Să se paralelizeze algoritmul secvențial implementat. Să se compare timpii de execuție între algoritmul secvențial și cel paralel și să se calculeze eficiența paralelă.

Algoritm secvențial

```
#include "stdafx.h"
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.h>

#define N 25000000
double a[N], b[N], c[N], x1[N], x2[N];

int main()

{
    clock_t start, stop;
    double durata_secv, durata_par;
    srand((unsigned)time(NULL));
    for (int i = 0; i < N; i++) {
        a[i] = (double)(rand() % 1000);
        b[i] = (double)(rand() % 1000);
        c[i] = (double)(rand() % 1000);
    }
    double delta;
    start = clock();
    for (int i = 0; i < N; i++) {
        delta = b[i] * b[i] - 4 * a[i] * c[i];
        if (delta >= 0) {
            x1[i] = (-b[i] + sqrt(delta)) / (2 * a[i]);
            x2[i] = (-b[i] - sqrt(delta)) / (2 * a[i]);
        }
    }
}
```

```
    }  
    stop = clock();  
    durata_secv = (double)(stop - start) / CLOCKS_PER_SEC;  
    printf("SECV %2.10f seconds\n", durata_secv);  
getchar();  
return 0;  
}
```

Algoritm paralel (varianta 1)

Paralelizarea unei probleme de calcul ce presupune executarea aceleiași operații pe un set de date multiplu (pe un vector) este foarte ușor de paralelizat deoarece gradul de paralelizare al algoritmului este egal cu dimensiunea setului de date (cu dimensiunea vectorului) iar partajarea de variabile între ramurile de execuție paralele se rezumă la indexul de parcurgere al setului de date – acești algoritmi se folosesc de paralelismul oferit de setul de date / paralelism de date.

Pentru implementarea paralelă vom duplica secțiunea de calcul și vom modifica instrucțiunea for după cum urmează (similar cu laboratorul precedent):

```
#define P 4  
#pragma omp parallel num_threads(P) shared(a, b, c, x1, x2)  
    private(delta)  
{  
    int ID = omp_get_thread_num();  
    for (int i = ID; i < N; i += P) {  
...  
    durata_par = (double)(stop - start) / CLOCKS_PER_SEC;  
    printf("P_V1 %2.10f seconds\n", durata_par);  
    printf("P_V1 FA=%2.2f\n", durata_secv/durata_par);  
    printf("P_V1 E(%d)=%2.2f\n", P, (durata_secv / durata_par) / P);  
}
```

În urma rulării programului vom observa o scădere a timpului de execuție (factor de accelerare supraunitar) dar o eficiență scăzută.

```
SECV 0.4320000000 seconds  
P_V1 0.2650000000 seconds  
P_V1 FA=1.63  
P_V1 E(4)=0.41
```

Modificați numărul de resurse paralele (task-uri) și vedeți cum variază eficiența paralelă.

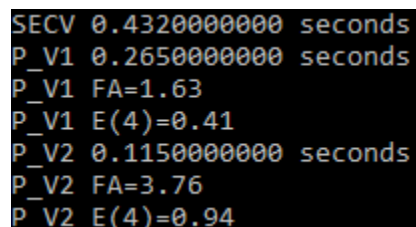
Algoritm paralel (varianta 2)

Biblioteca OpenMP dispune de o directivă specială de paralelizare a buclelor for ce permite automatizarea partajării indexului între firele de execuție paralele și optimizarea distribuirii efortului de calcul. Vom rescrie secțiunea for după cum urmează:

```
#pragma omp parallel for num_threads(P) shared(a, b, c, x1, x2)
                                private(delta)
    for (int i = 0; i < N; i++) {
...

```

În urma rulării programului se poate vedea o creștere a factorului de accelerare și implicit și a eficienței paralele:



```
SECV 0.4320000000 seconds
P_V1 0.2650000000 seconds
P_V1 FA=1.63
P_V1 E(4)=0.41
P_V2 0.1150000000 seconds
P_V2 FA=3.76
P_V2 E(4)=0.94
```

Algoritm paralel (varianta 3)

Directiva parallel for permite o clasă specială de clauze de paralelizare prin intermediul cărora putem controla procesul de distribuție a efortului de calcul. Aceste clauze sunt:

- schedule(static, chunk_size) – alocare statică (se face înaintea pornirii execuției) cu spargerea intervalului de calcul în bucăți de dimensiunea chunk_size;
- schedule(dynamic, chunk_size) – alocare dinamică (se face în momentul execuției) cu spargerea intervalului de calcul în bucăți de dimensiunea chunk_size;
- schedule(guided, chunk_size) – alocare dinamică cu spargerea intervalului de calcul în bucăți de dimensiunea variabilă (descrescător de la chunk_size la 1);
- schedule(runtime) – alocare dinamică cu planificare dinamică (stabilirea dimensiunii intervalului de calcul se face la execuție).

Vom testa următoarea formă a secțiunii paralele for:

```
#pragma omp parallel for num_threads(P) shared(a, b, c, x1, x2)
                                private(delta) schedule(static, 5)
    for (int i = 0; i < N; i++) {

```

și vom observa performanțe mai slabe chiar și față de prima variantă de implementare:

```
SECV 0.4320000000 seconds  
P_V1 0.2650000000 seconds  
P_V1 FA=1.63  
P_V1 E(4)=0.41  
P_V2 0.1150000000 seconds  
P_V2 FA=3.76  
P_V2 E(4)=0.94  
P_V3 0.2810000000 seconds  
P_V3 FA=1.54  
P_V3 E(4)=0.38
```

Din acest test putem concluziona că o alocare sau / și o planificare neinspirată pot conduce la scăderea dramatică a performanțelor algoritmului de calcul paralel.

Testați toate cele patru clauze de alocare / planificare și găsiți varianta cu cele mai bune performanțe (cât mai apropiate sau superioare variantei automate – varianta 2).