3. laboratorijska vježba AARSVP

Broj procesora Optimizacija	1	2	4	8	12	16
Sekvencijalno	17.74 s	16.6 s	16.2 s	16.186 s	15.7 s	15.15 s
For paralelizacija	18.36 s	8.88 s	5.48 s	3.63 s	3.025 s	2.67 s
Korištenje task	19.5 s	8.87 s	5.456 s	3.53 s	3.06 s	2.856 s
Nested paralelizacija	19.343 s	9.348 s	5.397 s	3.553 s	2.944 s	2.7486 s
Varijacija parametara (static)	15.44 s	8.92 s	5.4735 s	3.742 s	3.16 s	2.99 s
Varijacija parametara (dynamic)	18.82 s	21.5 s	9.95 s	6.21 s	4.807 s	4.34 s
Sve kombinirano zajedno	19 s	8.282 s	4.564 s	2.593 s	1.8655 s	1.8511 s

Može se primijetiti da svaka vrsta paralelizacije i razne kombinacije paralelizacija mogu, ali i ne moraju ubrzati program. Ako koristimo taskove, za dovoljno velik broj procesora će se pokazati poboljšanje. Ovisno o vrsti posla ponekad je bolje koristiti static, a ponekad dynamic schedule (u našem zadatku je static bolji zbog ujednačenosti količine posla). Isto vrijedi za ugniježđenu i “uobičajenu” paralelizaciju (u našem slučaju je klasična malo bolja). Najbolja opcija je pronaći neki kompromis sa svim ubrzanjima, to se događa u programu predanom u vježbi. Koristi se paralelizacija sa schedule(static) i collapse(2) te je optimizirano korištenje memorije, a critical je korišten samo u određenim situacijama.

U nastavku su prikazane implementacije korištene u kodu za neke pokušaje optimizacije.

Napomena: verzija korištena u drugoj laboratorijskoj vježbi pripada “For paralelizaciji”.

```
#pragma omp parallel
{
    #pragma omp single
    {
        for (int i = 0; i < HEIGHT; i += 2) {
            #pragma omp task firstprivate(i)
            {
                for (int j = 0; j < WIDTH; j += 2) {
                    int output_index = (i / 2) * (WIDTH / 2) + (j / 2);
                    int top_left = i * WIDTH + j;
                    int top_right = top_left + 1;
                    int bottom_left = top_left + WIDTH;
                    int bottom_right = bottom_left + 1;

                    U420[output_index] = (U444[top_left] + U444[top_right] +
                                          U444[bottom_left] + U444[bottom_right]) / 4;
                    V420[output_index] = (V444[top_left] + V444[top_right] +
                                          V444[bottom_left] + V444[bottom_right]) / 4;
                }
            }
        }
    }
}
```

Korištenje task-ova

```
#pragma omp parallel for schedule(static)
for (int i = 0; i < FRAME_SIZE; i++) {
    Y[i] = (uint8_t)(0.257 * R[i] + 0.504 * G[i] + 0.098 * B[i] + 16);
    U[i] = (uint8_t)(-0.148 * R[i] - 0.291 * G[i] + 0.439 * B[i] + 128);
    V[i] = (uint8_t)(0.439 * R[i] - 0.368 * G[i] - 0.071 * B[i] + 128);
}
```

Varijacija parametara (static)

```
#pragma omp parallel for schedule(dynamic, 8)
for (int i = 0; i < HEIGHT; i += 2) {
    for (int j = 0; j < WIDTH; j += 2) {
        int output_index = (i / 2) * (WIDTH / 2) + (j / 2);
        int top_left = i * WIDTH + j;
        int top_right = i * WIDTH + (j + 1);
        int bottom_left = (i + 1) * WIDTH + j;
        int bottom_right = (i + 1) * WIDTH + (j + 1);

        U420[output_index] = (U444[top_left] + U444[top_right] + U444[bottom_left] + U444[bottom_right]) / 4;
        V420[output_index] = (V444[top_left] + V444[top_right] + V444[bottom_left] + V444[bottom_right]) / 4;
    }
}
```

Varijacija parametara (dynamic)

```
#pragma omp parallel for
for (int i = 0; i < HEIGHT; ++i) {
    #pragma omp parallel for
    for (int j = 0; j < WIDTH; ++j) {
        int position420 = (i / 2) * (WIDTH / 2) + (j / 2);
        int position444 = i * WIDTH + j;
        U444[position444] = U420[position420];
        V444[position444] = V420[position420];
    }
}
```

Ugniježđeni paralelizam

```
#pragma omp parallel for
for (int i = 0; i < HEIGHT; ++i) {
    for (int j = 0; j < WIDTH; ++j) {
        int position420 = (i / 2) * (WIDTH / 2) + (j / 2);
        int position444 = i * WIDTH + j;
        U444[position444] = U420[position420];
        V444[position444] = V420[position420];
    }
}
```

Uobičajeni paralelizam