گزارش کار تمرین کامپیوتری ششم

درس برنامه نویسی موازی

رستا تدین- 810196436

دیار محمدی- 810196553

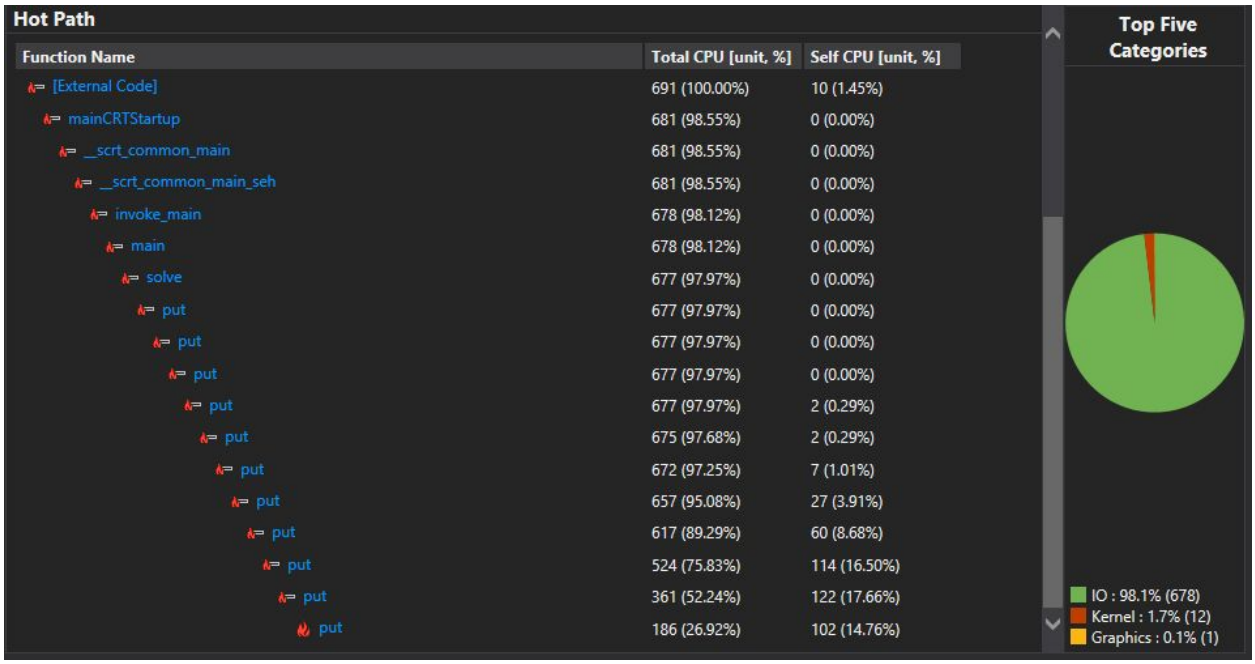
توضیح کد سریال

کد سریال N Queens یک کد بازگشتی است که با قرار دادن وزیرها در خانه های مختلف شطرنج و backtrack کردن قصد پیدا کردن تعداد راه هایی که می توان در یک صفحه N*N شطرنجی N وزیر قرار داد را به دست آورد. این کد از 2 تابع اصلی ساخته شده است. تابع solve و تابع put. در تابع put در صورتی که در خانه فعلی یک وزیر قرار دهیم و توسط وزیرهای دیگر مورد حمله قرار نگیرد این کار را می کند در غیر این صورت return کرده و backtrack می کند. اگر به سطر آخر رسیده باشیم یکی به تعداد جواب های پیدا شده اضافه می شود و در غیر این صورت سطر را یکی زیاد می کند.

در تابع solve نیز به تعداد N بار با شروع از سطر نخست و ستون i ام (یعنی قرار دادن در سطر اول و ستون i ام) کار را شروع کرده و تابع put را صدا زده تا به این ترتیب تمام پاسخ ها پیدا شوند.

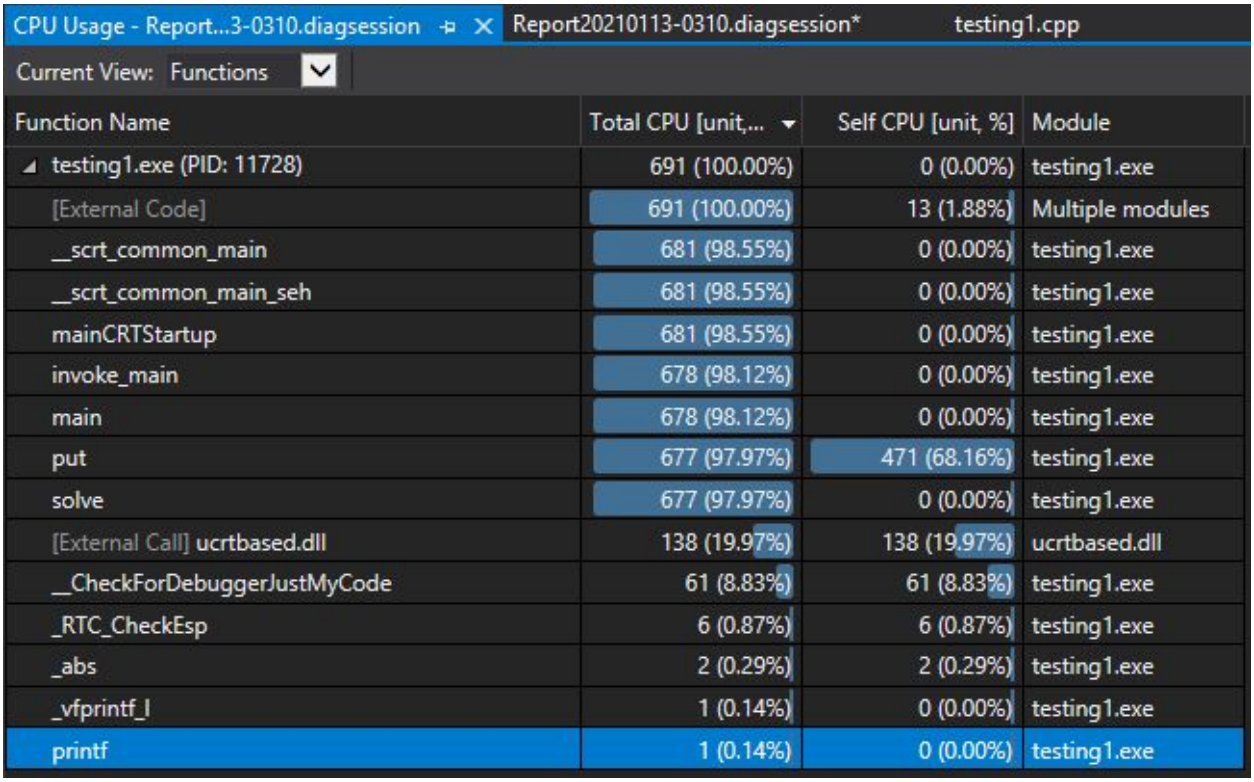
پیدا کردن Hot Spot ها

در این تمرین کامپیوتری قصد داریم مسئله N Queens را با استفاده از intel parallel studio موازی سازی کرده و زمان اجرا را کاهش دهیم. به این منظور ابتدا کد serial را بررسی کرده و hotspotهای آن را به دست می آوریم. با استفاده از ابزار performance profiler نرم افزار visual studio برای hotpath به نتیجه زیر رسیدیم:



تابعی که hotspot است تابع put است که به صورت بازگشتی خود را صدا می زند. همچنین در مقیاس دیگری مقدار استفاده CPU برای هر یک از توابع را مشاهده می کنیم. مشاهده می شود تابع put که مسئولیت گذاشتن

وزیرها در خانه های مختلف و شمردن تعداد جواب ها را دارد زمان بیشتری از همه توابع CPU را درگیر می کند.



CPU Usage - Report...3-0310.diagsession | Report20210113-0310.diagsession* | testing1.cpp

Current View: Functions

Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
testing1.exe (PID: 11728)	691 (100.00%)	0 (0.00%)	testing1.exe
[External Code]	691 (100.00%)	13 (1.88%)	Multiple modules
__scrt_common_main	681 (98.55%)	0 (0.00%)	testing1.exe
__scrt_common_main_seh	681 (98.55%)	0 (0.00%)	testing1.exe
mainCRTStartup	681 (98.55%)	0 (0.00%)	testing1.exe
invoke_main	678 (98.12%)	0 (0.00%)	testing1.exe
main	678 (98.12%)	0 (0.00%)	testing1.exe
put	677 (97.97%)	471 (68.16%)	testing1.exe
solve	677 (97.97%)	0 (0.00%)	testing1.exe
[External Call] ucrtbased.dll	138 (19.97%)	138 (19.97%)	ucrtbased.dll
__CheckForDebuggerJustMyCode	61 (8.83%)	61 (8.83%)	testing1.exe
_RTC_CheckEsp	6 (0.87%)	6 (0.87%)	testing1.exe
_abs	2 (0.29%)	2 (0.29%)	testing1.exe
_vfprintf_l	1 (0.14%)	0 (0.00%)	testing1.exe
printf	1 (0.14%)	0 (0.00%)	testing1.exe

تابع put به صورت زیر است:

```
int put(int Queens[], int row, int column)
{
    int i;
    for (i = 0; i < row; i++) {
        if (Queens[i] == column || abs(Queens[i] - column) == (row - i))
            return -1;
    }
    Queens[row] = column;
    if (row == N - 1) {
        solutions++;
    }
    else {
        for (i = 0; i < N; i++) { // increment row
            put(Queens, row + 1, i);
        }
    }
    return 0;
}
```

با توجه به اینکه در تابع put تابع abs به تعداد row بار تکرار شده و صدا زده می شود در نهایت این تابع بیشتر از هر تابعی در این برنامه صدا زده می شود.

موازی سازی اولیه

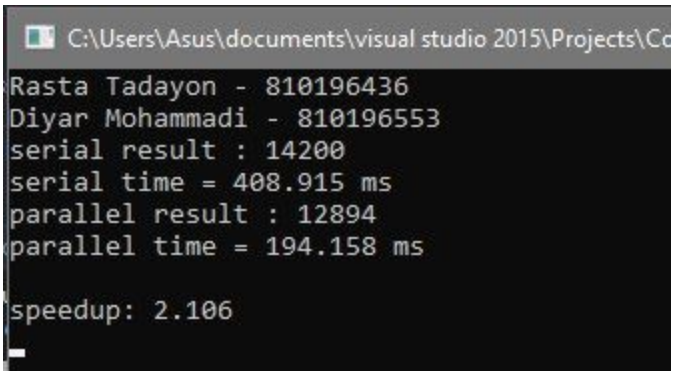
برای مرحله نخست موازی سازی تابع solve را به صورت موازی در می آوریم. برای این کار از #pragma omp for استفاده می کنیم و قسمت های مختلف حلقه for را به thread های مختلف نسبت می دهیم. برای اضافه کردن تعداد جواب ها نیز برای جلوگیری از اتفاق افتادن data race از atomic استفاده می کنیم. به این ترتیب تابع solve به صورت زیر در می آید:

```
void solve_parallel() {
    int tid;
    #pragma omp parallel for private(tid)
    for (int i = 0; i < N; i++) {
        tid = omp_get_thread_num();
        int* queens = new int[N]; // Array of queens on the chess board.

        put_parallel(queens, 0, i, tid);

        delete[] queens;
    }
```

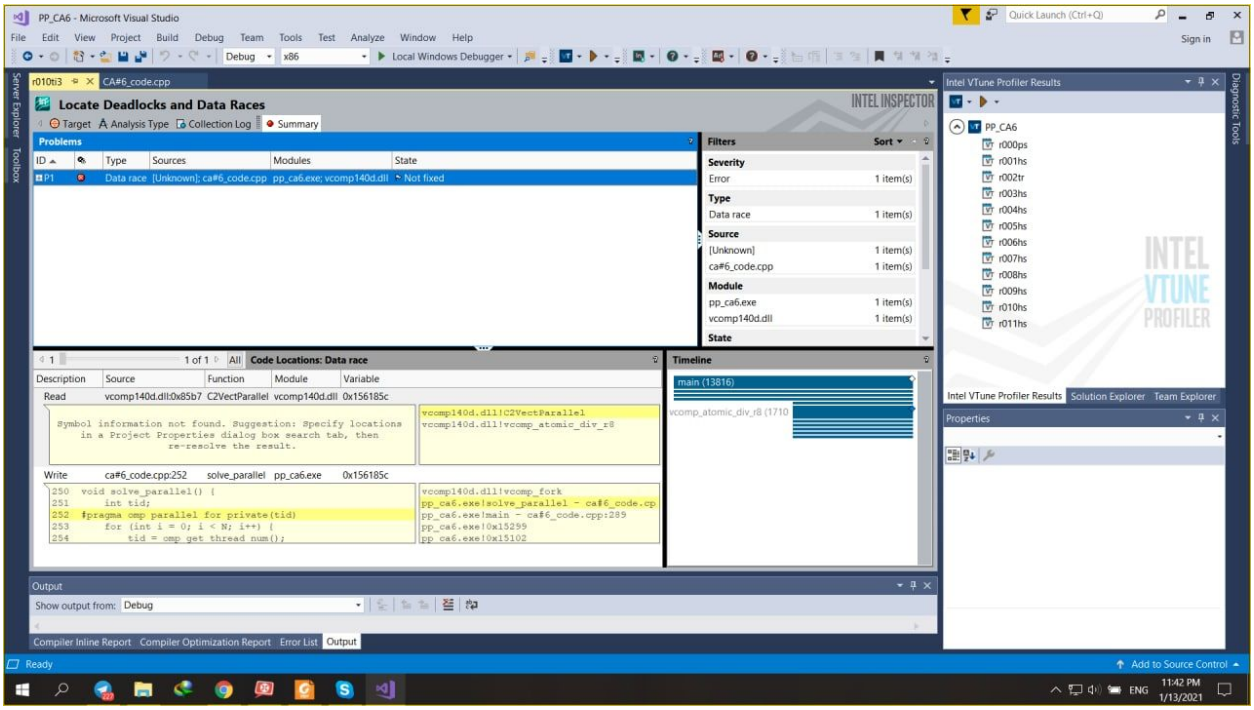
همچنین تابع put نیز با تغییرات جزئی به صورت زیر در می آید:

```
void put_parallel(int queens[], int row, int col, int tid) {

    for (int i = 0; i < row; i++) {
        if ((queens[i] == col) || (abs(queens[i] - col) == (row - i))) {
            return;
        }
    }

    queens[row] = col;
    if (row == (N - 1)) {
#pragma omp atomic

        solutions_P++;
    }
    else {
        for (int i = 0; i < N; i++) {
            put_parallel(queens, row + 1, i, tid);
        }
    }
}
```

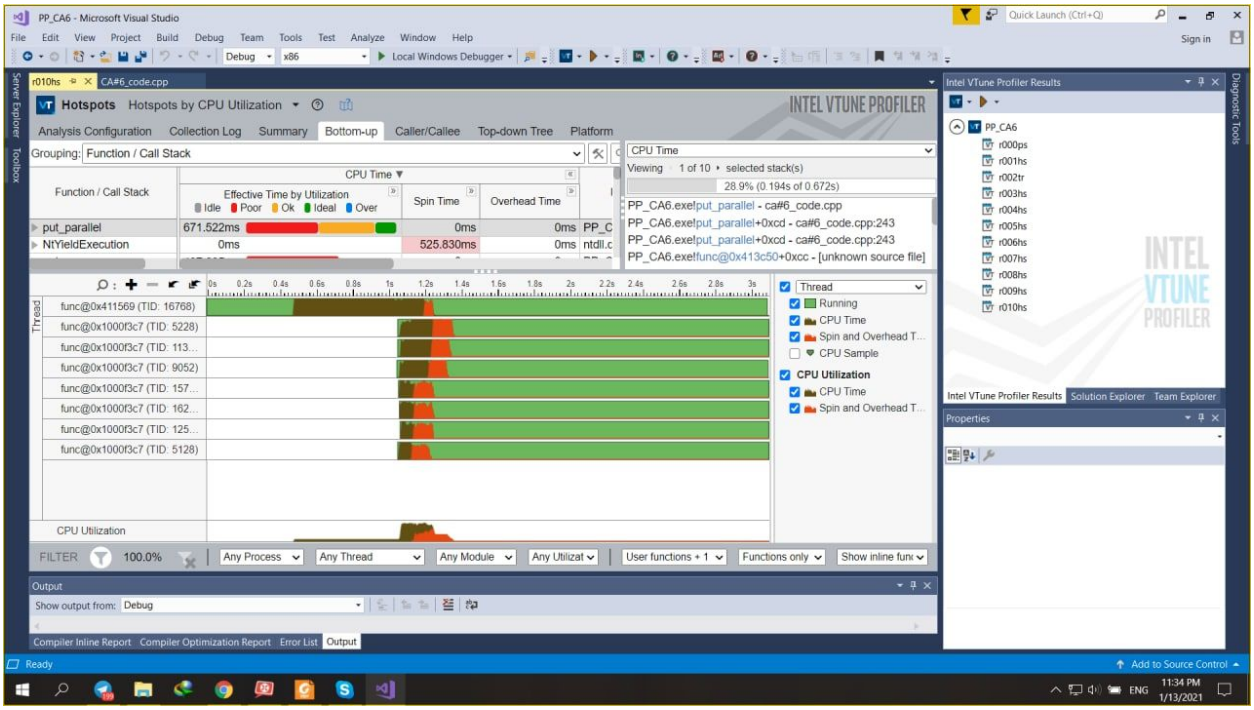
مقدار speedup در این حالت:



سپس برنامه را با Inspector بررسی میکنیم.

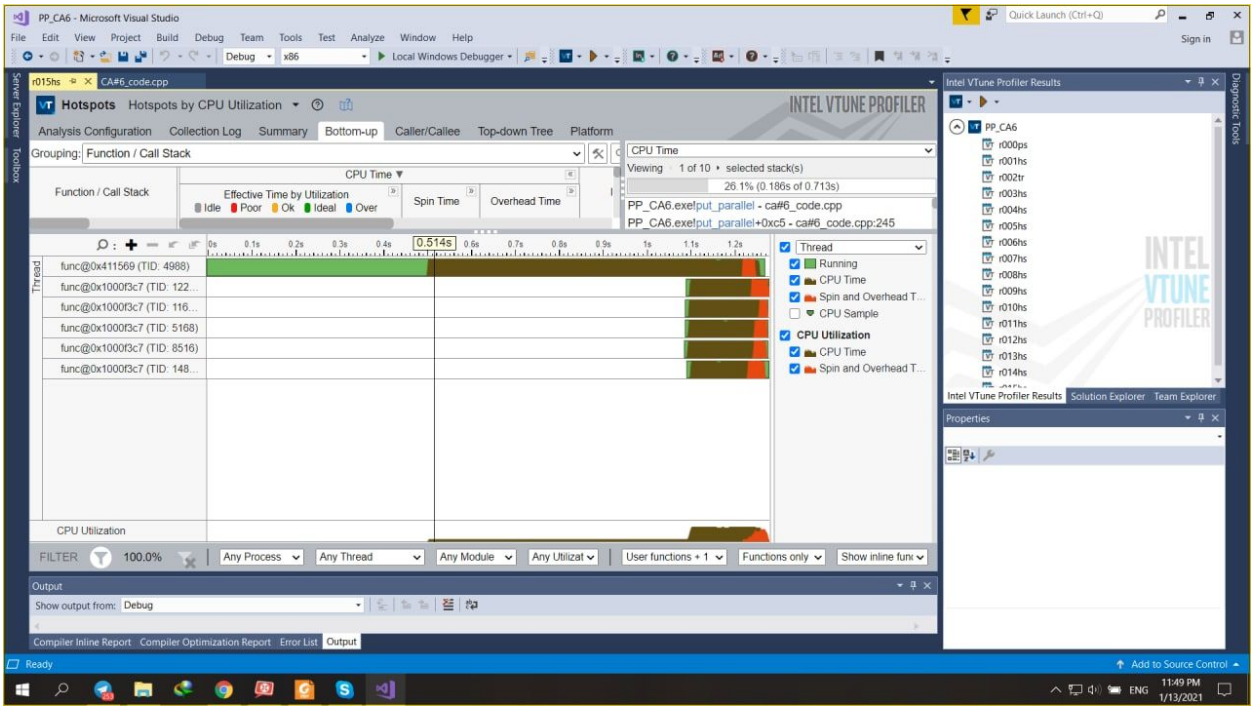


و میبینیم که data race وجود ندارد.

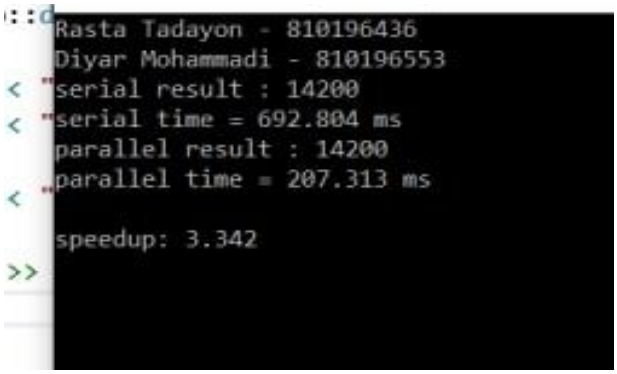


در تصویر بالا می بینیم که با تعداد ۸ ترد توازن بین ترد ها به خوبی حفظ نمی شود چون تعداد مسائل ۱۲ تا است و ۴ ترد اول هر کدام ۲ مساله برای حل و ۴ ترد دیگر تنها یک مساله برای حل دارند.

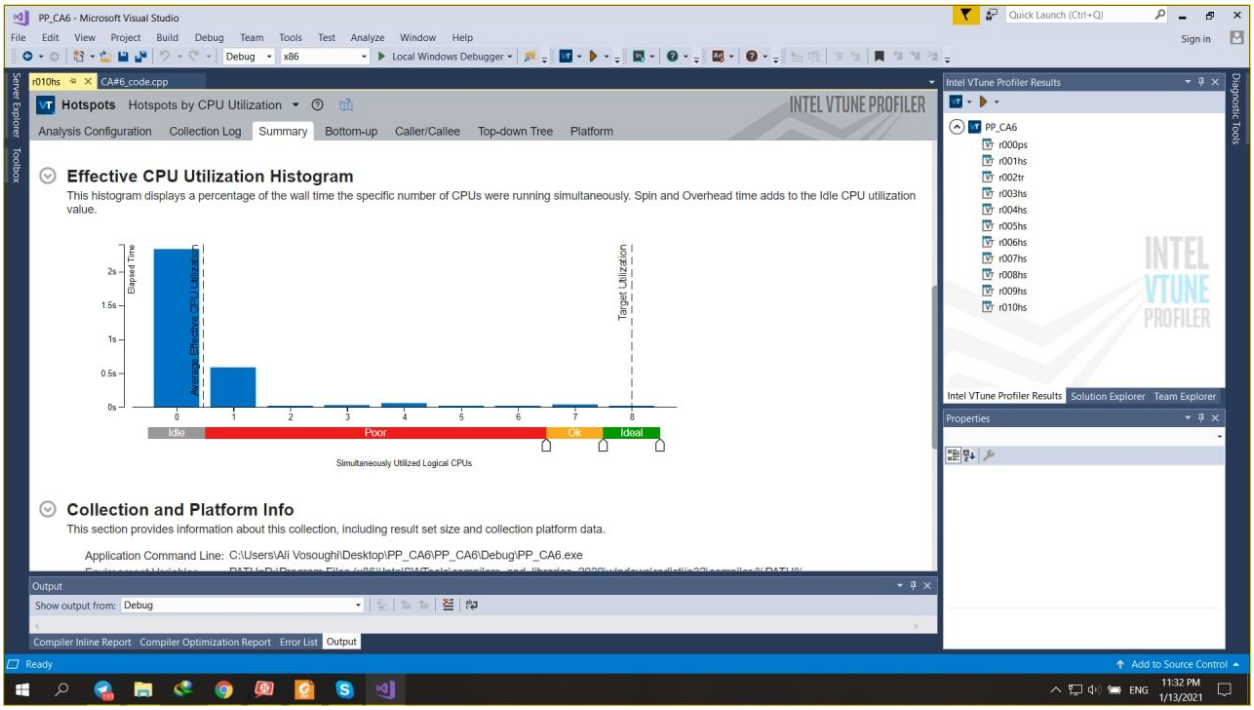
برای حل این مشکل تعداد نردها را ۶ قرار میدهیم که توازن برقرار باشد.



می‌بینیم که توازن کاملا برقرار است و زمان اجرا به شکل زیر است.



در حالت هشت ترد داریم:



که نشان می‌دهد زمان زیادی cpu کاری نمیکرده، نیم ثانیه تنها یک ترد کار میکرده که مربوط به حالت سریال و همچنین زمان ساختن تردهای حالت موازی است و مقدار زمان بسیار کمی هر ۸ ترد با هم کار کرده اند.

در حالت ۶ ترد میبینیم که زمان بسیار زیادی نسبت به قبل تمام ۶ ترد درگیر مشغول کار بوده‌اند که یعنی tune کردن ما هر ۶ ترد را با هم به کار گرفته است.