

پروژه چهارم برنامه نویسی موازی

کسرانوربخش ۸۱۰۱۰۰۲۳۰

شهنام فیضیان ۸۱۰۱۰۰۱۹۷

مقدمه

در این تمرین به کار با ابزار های معرفی شده در درس یعنی: Intel Vtune Profiler, Intel Advisor, Intel Inspector پرداختیم. در بخش اول، مساله k اسب در صفحه شطرنج را بهبود بخوبیم، از آنجایی که کد سریال اولیه تمیز بود و کامنت های خوبی هم برای هر فانکشن اش داشت به توضیح این که هرتابع چه کاری را انجام می دهد، نپرداختیم. در بخش دوم هم، تمامی مشکلات مربوط به حافظه کد داده شده را پیدا و اصلاح کردیم، همچنین از لحاظ static security هم کد بررسی و اصلاح گردید. ضمناً تمامی تحلیل های گرفته شده در دو پوشه: part1 reports و part2 reports موجود اند.

بخش اول(کد علیرضا)

در ابتدا شروع به کار مشکلی با متغیر count وجود داشت که آن را میهم می خواند که تغییر نام به num_solution دادیم. برای محاسبه زمان اجرای برنامه هم ابتدا از timeGetTime() که در کتابخانه winmm می باشد استفاده کردیم. با تست کردن 3 کامپایلر مختلف Intel C++, Intel OneAPI DPC++, Visual Studio Compiler به نتایج تقریباً مشابهی در حدود 8 تا 12 میلی ثانیه برای اجرای برنامه رسیدیم.

	Code	Description
abc	E0266	"count" is ambiguous
abc	E0266	"count" is ambiguous
x	C2872	'count': ambiguous symbol
x	C2872	'count': ambiguous symbol

عکس 1: مشکل متغیر count

```
starttime = timeGetTime();
/* Creation of a m*n board */
char** board = new char* [m];
for (int i = 0; i < m; i++) {
    board[i] = new char[n];
}

/* Make all the places are empty */
makeBoard(board);

kkn(k, 0, 0, board);

elapsedtime = timeGetTime() - starttime;
printf("Time Elapsed %2d mSecs Total number of solutions : = %d\n", (int)elapsedtime, num_solution);
```

قطعه کد ۱: اضافه کردن کد محاسبه زمان اجرای برنامه و تغییر نحوه خروجی دادن(بیش از این یک cout را داشتم که فقط پاسخ را چاپ می کرد)

```

K   K   K
A   A   A
A   A   A
K   K   K

K   A   K
A   K   A
K   A   K
A   K   A

A   K   A
K   A   K
A   K   A
K   A   K

Time Elapsed  9 mSecs Total number of solutions : = 3

```

عکس 2: نمونه ای از خروجی برنامه با Intel C++ Compiler

به سراج محاسبه دقیق تر زمان اجرای برنامه رفتیم با استفاده از QueryPerformanceCounter

```

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);
/* Creation of a m*n board */
char** board = new char* [m];
for (int i = 0; i < m; i++) {
    board[i] = new char[n];
}

/* Make all the places are empty */
makeBoard(board);

kkn(k, 0, 0, board);
QueryPerformanceCounter(&finish);

elapsedTime = static_cast<double>(finish.QuadPart - start.QuadPart) / frequency.QuadPart;

cout << "Elapsed time: " << elapsedTime << " seconds |" << " Total number of solutions: " << num_solution << endl;

```

قطعه کد 2: دقیق تر کردن نحوه محاسبه زمان اجرای برنامه

```

K   K   K
A   A   A
A   A   A
K   K   K

K   A   K
A   K   A
K   A   K
A   K   A

A   K   A
K   A   K
A   K   A
K   A   K

Elapsed time: 0.0099009 seconds  Total number of solutions: 3

```

عکس 3: نمونه ای از خروجی برنامه که باز هم حدود 9 تا 10 میلی ثانیه می باشد

حال به سراغ optimization های موجود رفتیم، با استفاده از O1/, O2/, O3/

نتایج 5 بار اجرا بدون هیچ بهینه سازی:

Elapsed time: 0.009908 seconds Total number of solutions: 3

Elapsed time: 0.0067732 seconds Total number of solutions: 3

Elapsed time: 0.0100206 seconds Total number of solutions: 3

Elapsed time: 0.0099959 seconds Total number of solutions: 3

Elapsed time: 0.0110341 seconds Total number of solutions: 3

نتایج 5 بار اجرا با O1 که تمرکز بر سایز دارد:

Elapsed time: 0.0100874 seconds Total number of solutions: 3

Elapsed time: 0.0061856 seconds Total number of solutions: 3

Elapsed time: 0.0087874 seconds Total number of solutions: 3

Elapsed time: 0.0058814 seconds Total number of solutions: 3

Elapsed time: 0.0060019 seconds Total number of solutions: 3

نتایج 5 بار اجرا با O2 که تمرکز بر سرعت دارد:

Elapsed time: 0.0081414 seconds Total number of solutions: 3

Elapsed time: 0.0089925 seconds Total number of solutions: 3

Elapsed time: 0.0093125 seconds Total number of solutions: 3

Elapsed time: 0.0109234 seconds Total number of solutions: 3

Elapsed time: 0.0094353 seconds Total number of solutions: 3

نتایج 5 بار اجرا با O3 که بیشترین بهینه سازی است:

Elapsed time: 0.0091415 seconds Total number of solutions: 3

Elapsed time: 0.0055399 seconds Total number of solutions: 3

Elapsed time: 0.0065087 seconds Total number of solutions: 3

Elapsed time: 0.0077862 seconds Total number of solutions: 3

Elapsed time: 0.0061308 seconds Total number of solutions: 3

چون خروجی ها تعداد زیادی داشتند، در اینجا آورده نشد اما چک گردید که در تمامی حالات خروجی درست داده شود و سپس هم این ها disable شدند تا بتوانیم مقایسه درستی از حالت سریال و موازی داشته باشیم.

بخش اول، آنالیز:

به صورت چشمی وقتی به کد پرورژه می نگریم تابع های استفاده شده به جز `kkn` آنقدر زمان بر نیستند چون اغلب از مرتبه زمانی 2^{18} هستند و دستور یا فراخوانی زمان بری ندارند و به نظر میرسد که موازی کردن `kkn` می تواند خوب باشد همچنان چون هیچ یک از تابع های دستور محاسباتی به خصوصی ندارند و اینکه کار با `I/O` هم داریم، **Vectorization** خاصی را نمی توان انجام داد.

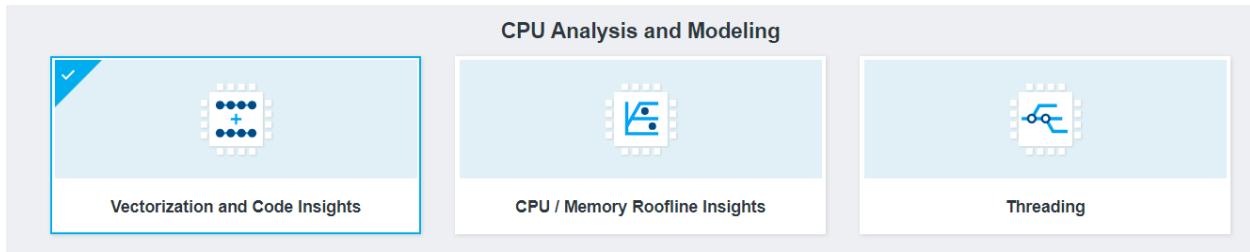
همان طور که در خروجی های قبل مشاهده شد، مدت زمان اجرای برنامه بسیار کم است و وقتی می خواهیم با Advisor کار بکنیم، داده کافی را ندارد پس صفحه مان را 4 در 4 در نظر می گیریم و تعداد اسپ هارا 3 می گذاریم:

Elapsed time: 0.948561 seconds Total number of solutions: 276

ابتدا از ابزار Intel Advisor می خواهیم استفاده کنیم. این ابزار قابلیت هایی دارد که می توان در این لینک:

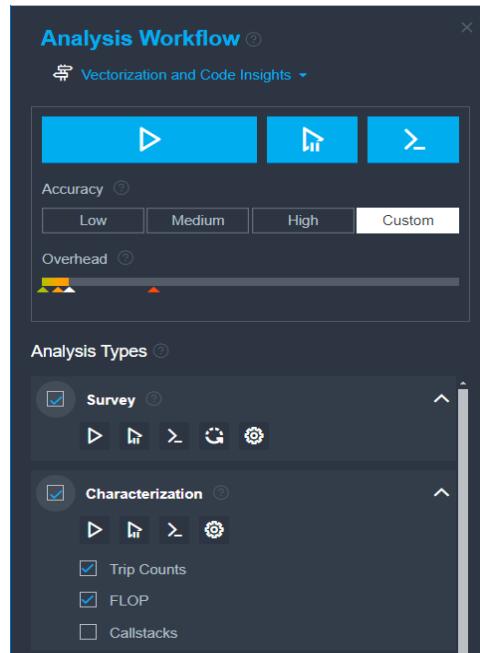
<https://www.intel.com/content/www/us/en/docs/advisor/2023-0/overview.html>

خلاصه ای از آن ها را دید که به کار ما بیشتر این 3 نوع تحلیل می آیند:



عکس 4: انواع بررسی ها روی CPU

ابتدا از Analysis Workflow با این پارامتر ها:



عکس 5: نحوه ریپورت گرفتن از Advisor

استفاده کردیم که در واقع مشخص می‌کند چه مقدار از زمان در برنامه در کدام loop ها است و مشکلات احتمالی که جلوی خوب vectorization را بگیرد می‌دهد. در Summary این ریپورت بخش‌های مفیدی هست:

Top Time-Consuming Loops			
Loop	Self Time	Total Time	Trip Counts
loop in kkn at Alireza's-Code.cpp:120	<0.001s	0.292s	1
loop in kkn at Alireza's-Code.cpp:121	<0.001s	0.292s	3
loop in displayBoard at Alireza's-Code.cpp:29	<0.001s	0.286s	4
loop in displayBoard at Alireza's-Code.cpp:30	<0.001s	0.251s	4

عکس 6: حلقه‌های زمان بر

Recommendations	
! Remove system function call(s) inside loop	loop in displayBoard at Alireza's-Code.cpp:30
! Vectorize call(s) to virtual method	loop in displayBoard at Alireza's-Code.cpp:30

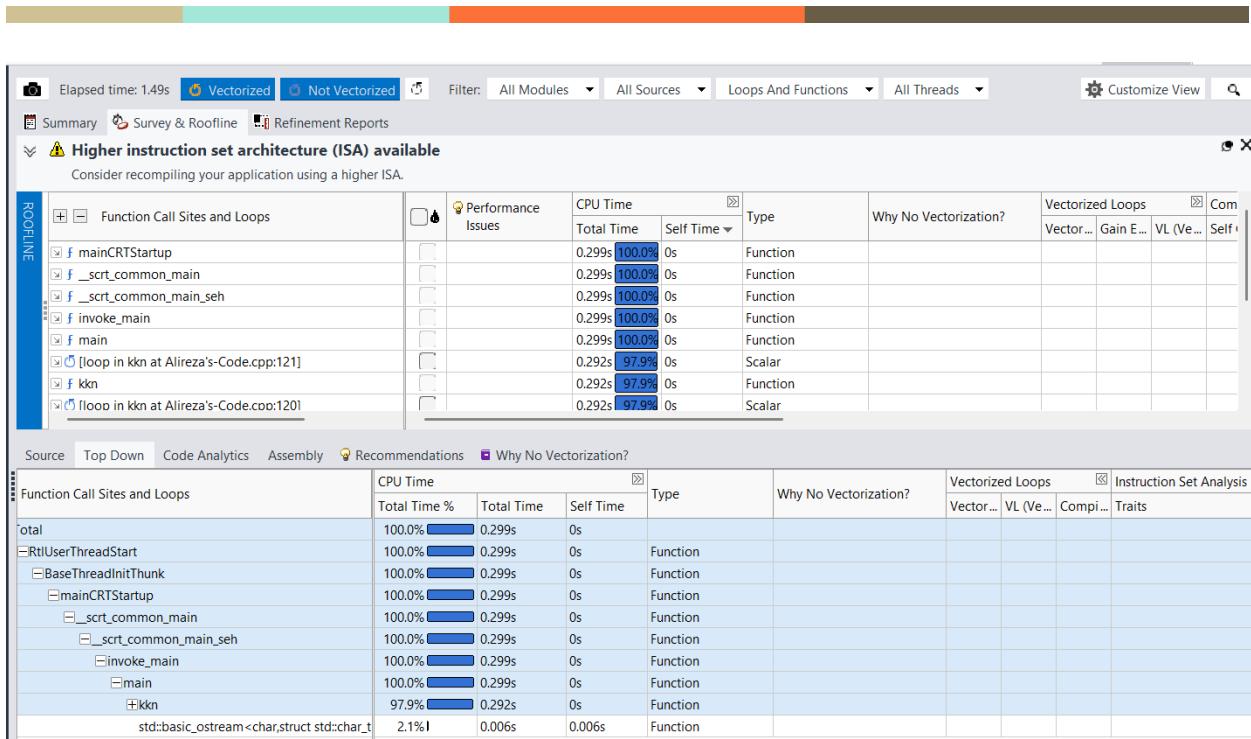
عکس 7: پیشنهاد‌ها برای بهبود برنامه

مثلاً در اینجا می‌گوید چون که داخل لوب چاپ کردن صفحه برنامه، داریم از تابع cout سیستم یعنی cout استفاده می‌کنیم امکان وجود ندارد که خوب نیست و بهتر است که حذف شود که در ادامه می‌بینیم که از تکنیک بافر کردن استفاده کردیم.

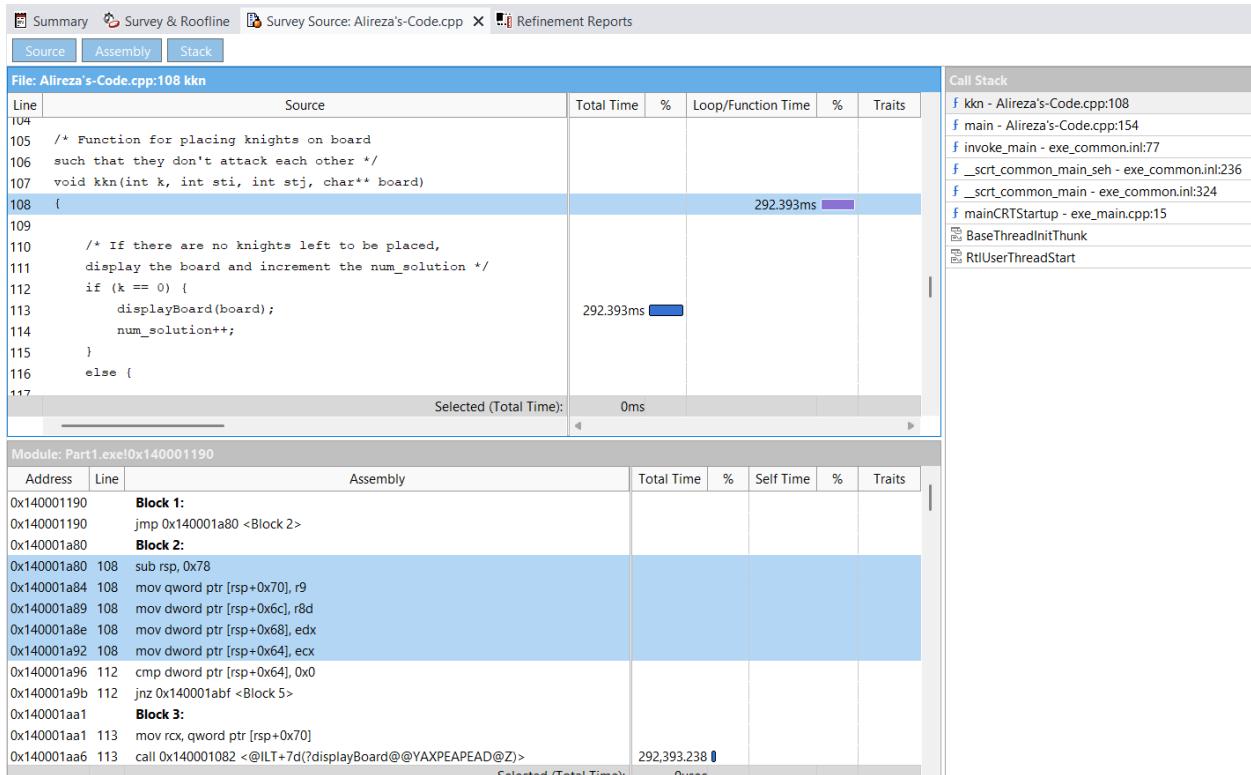
Platform Information	
CPU Name	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Frequency	1.80 GHz
Logical CPU Count	8
Operating System	Windows
Computer Name	Kasra

عکس 8: سخت افزاری که برنامه روی آن دارد اجرا می‌شود.

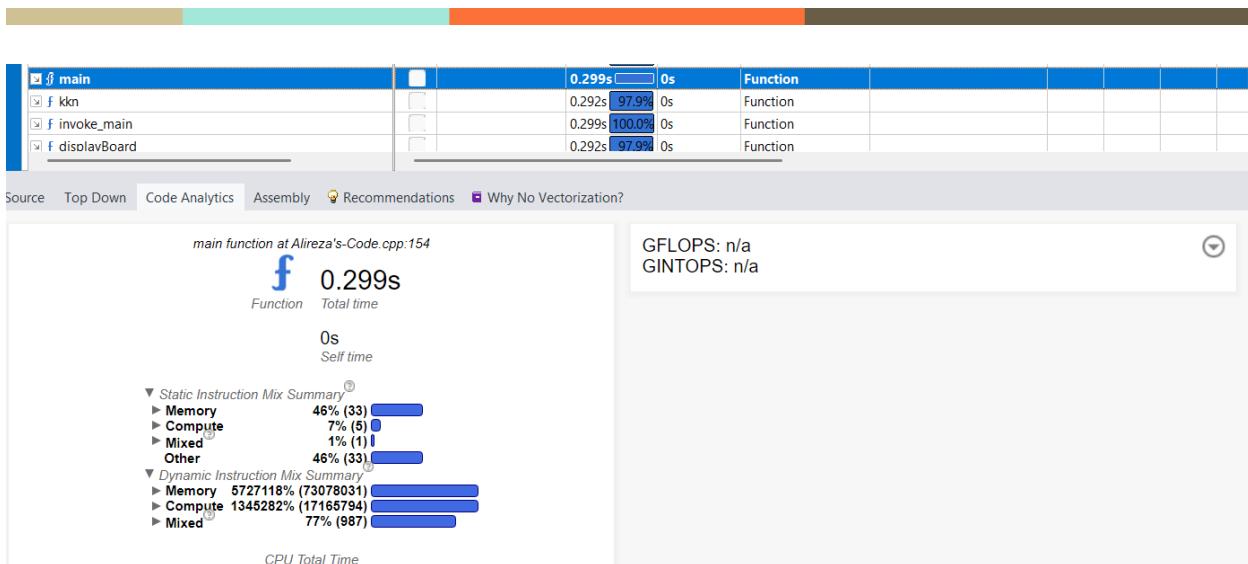
همانطور که حدس می‌زدیم تابع kkn است که تایم زیادی از اجرای برنامه را دارد به خودش اختصاص می‌دهد، همچنین تابع display board چون با هر بار قرار دادن یک اسپ صدا می‌شود و باید کل صفحه را بکشد پس این دو می‌توانند کاندید های خوبی برای موازی سازی باشند اما چون ما در display board تابع کار با o/a را داریم که thread safe نیست، آن چنان هم کاندید خوبی به حساب نمی‌آید.



عکس 9: خروجی گزارش داده شده که در آن مبینیم kkn چقدر زمان بر بوده است.

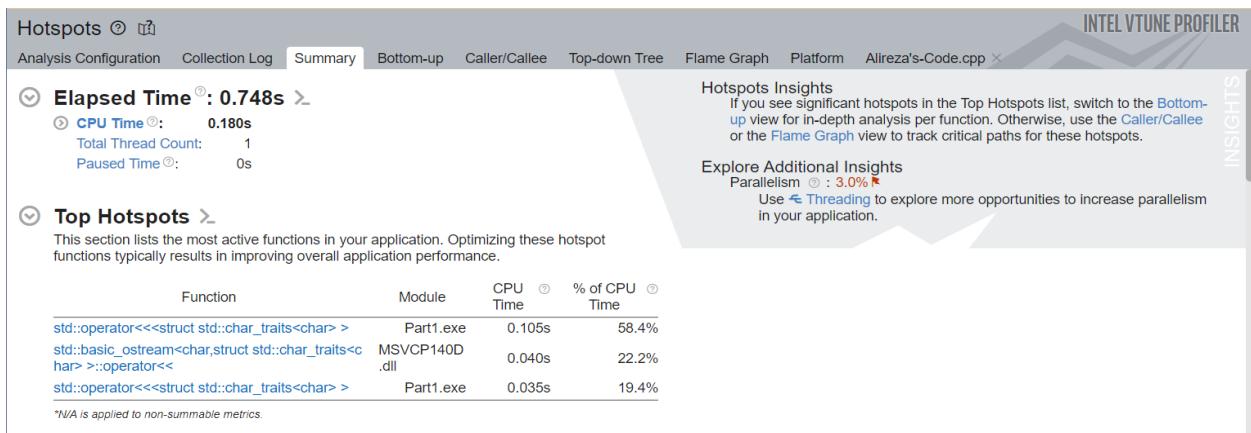


عکس 10: اطلاعات بیشتری از این تابع و حتی کد اسembلی و استک آن

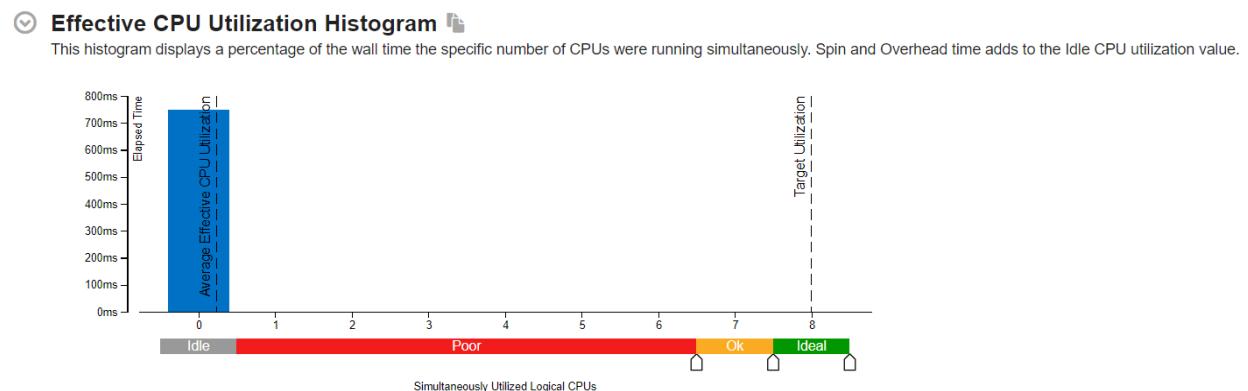


عکس 11: میزان زمان صرف شده در main که همانطور که میدانیم بخش اعظم اش کار با حافظه است

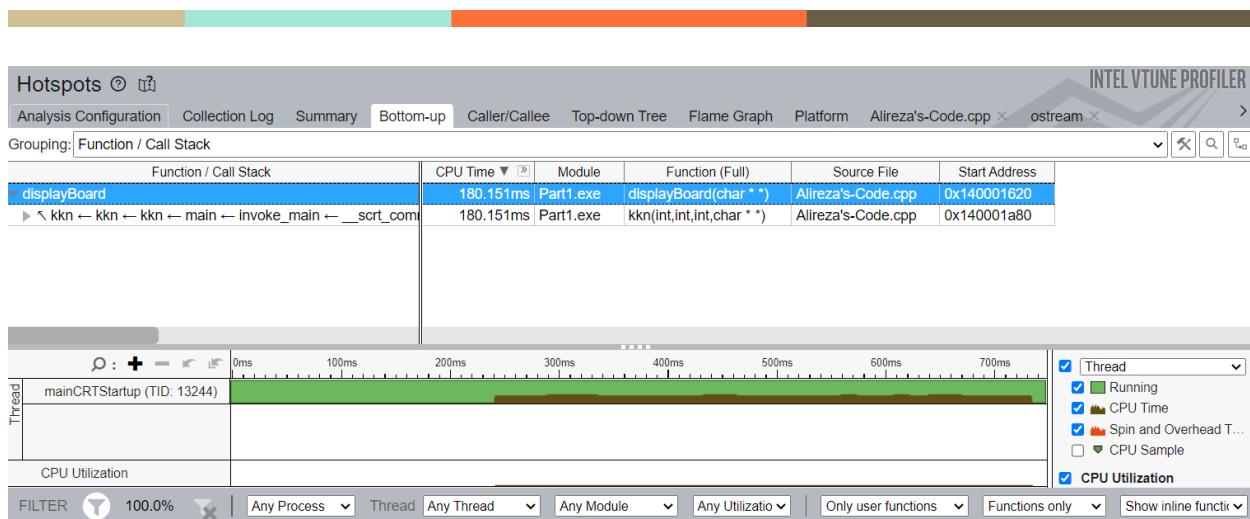
حال به سراغ VTune Profiler می رویم تا hotspot های برنامه را از طریق آن هم بررسی کرده باشیم.



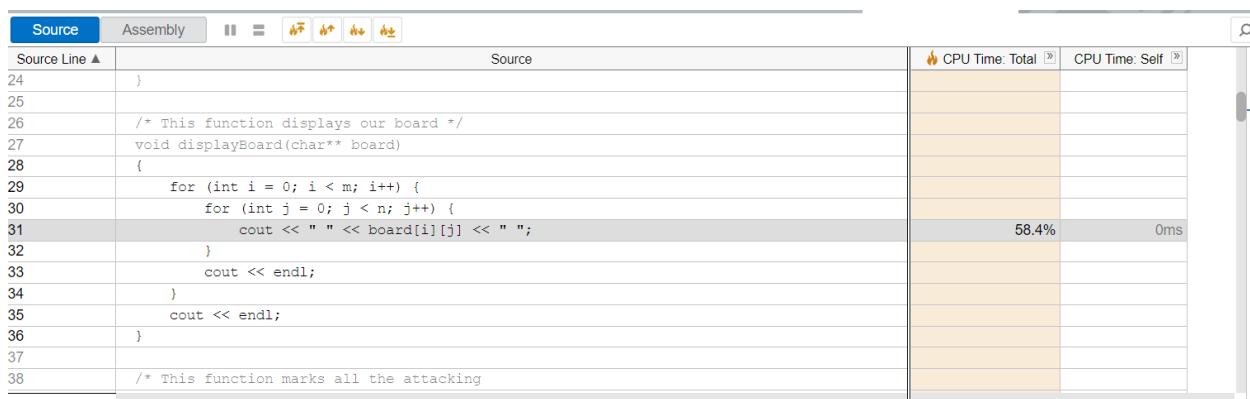
عکس 12: خلاصه گزارش تحلیل hotspot



عکس 13: ادامه عکس 12



عکس 14: باز هم همانطور که حدس میزدیم و دیدیم، displayBoard دارد زمان می‌گیرد



عکس 15: در اینجا هم مثهود است

در انتهای این بخش نتایج 5 بار اجرای سریال برنامه را ذخیره می‌کنیم:

Elapsed time: 2.36343 seconds Total number of solutions: 276

Elapsed time: 2.29598 seconds Total number of solutions: 276

Elapsed time: 2.32634 seconds Total number of solutions: 276

Elapsed time: 2.28778 seconds Total number of solutions: 276

Elapsed time: 2.35232 seconds Total number of solutions: 276

بخش دوم، پیاده سازی:

همان طور که در بخش اول دیدیم، بخشی از برنامه ما دارد زمان زیادی می‌گیرد که به صورت بازگشتی است. با جست و جو در این لینک:

<https://stackoverflow.com/questions/76276934/looking-for-a-good-way-to-use-openmp->

نمونه ای از برنامه فیبوناچی که بازگشتی است و با استفاده از OpenMP موازی شده است را دیدیم. با

دیدن این لینک هم: <https://www.ibm.com/docs/en/zos/2.4.0?topic=processing-pragma-omp-task> فهمیدیم که با ساختار task می توان برنامه های بازگشتی را موازی کرد. ابتدا به همین سمت رفتیم: برای گام اول الگوریتم باید با یک ترد اجرا شود:

```
#pragma omp parallel
```

```
{
```

```
#pragma omp single
```

```
kkn(k, 0, 0, board);
```

```
}
```

و سپس داخل تابع kkn برای فرآخوانی displayBoard از #pragma omp critical استفاده کردیم که صفحه بازی که به درستی چاپ شود و تعداد راه حل های پیدا شده هم صحیح باشد. برای هر فرآخوانی بازگشتی kkn هم یک تک spawn کردیم. اما این پیاده سازی درست کار نکرد و خروجی صحیحی از نظر تعداد جواب ها نداد. فهمیدیم برای این که درست شود باید متغیر های i, j, k و همچنین صفحه بازی جدید را یعنی new_board را به صورت firstprivate تعریف کنیم که هر task ای که ایجاد شود، صفحه خودش را در حین spawn شدن داشته باشد.

سپس جدگانه به سراغ موازی سازی hot spot اصلی برنامه یعنی، تابع displayBoard رفتیم چون به همراه یکدیگر درست کار نکرند، در اینجا با استفاده از بافر کردن و critical section از عملکرد درست آن اطمینان حاصل می کنیم:

```
void displayBoard(char** board)

    string output = ""; // Buffer to store output

# pragma omp parallel for

for (int i = 0; i < m; i++) {

    string local_output = ""; // Private output for each thread

    for (int j = 0; j < n; j++) {

        local_output += " " + string(1, board[i][j]);" " +
    }

    local_output += "\n";

#     pragma omp critical

    {
        output += local_output;
    }

}

cout << output << endl;
```

قبل از این، برای آن که مقایسه درستی از نسخه سریال و موازی داشته باشیم، نسخه سریالمان را به روز می کنیم یعنی تابع در `displayBoard` آن هم از تکنیک بافر کردن استفاده می کنیم:

```
void displayBoard(char** board) {
    string output = "";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            output += " " + string(1, board[i][j]) + " ";
        }
        output += "\n";
    }
    output += "\n";
    cout << output;
}
```

در این بخش نتایج 5 بار اجرای برنامه سریال بهبود یافته را ذخیره می کنیم:

Elapsed time: 0.403985 seconds Total number of solutions: 276

Elapsed time: 0.287522 seconds Total number of solutions: 276

Elapsed time: 0.293897 seconds Total number of solutions: 276

Elapsed time: 0.261661 seconds Total number of solutions: 276

Elapsed time: 0.358212 seconds Total number of solutions: 276

در این بخش نتایج 5 بار اجرای برنامه موازی شده تابع `kkn` را ذخیره می کنیم(که مطابق با پیش بینی چون داریم از استفاده می کنیم، طولانی تر شد):

Elapsed time: 0.334722 seconds Total number of solutions: 276

Elapsed time: 0.309563 seconds Total number of solutions: 276

Elapsed time: 0.336259 seconds Total number of solutions: 276

Elapsed time: 0.322832 seconds Total number of solutions: 276

Elapsed time: 0.313084 seconds Total number of solutions: 276

در این بخش نتایج 5 بار اجرای برنامه موازی شده `displayBoard` را ذخیره می کنیم(این جا هم چون را داریم باز کند تر شد):

Elapsed time: 0.356329 seconds Total number of solutions: 276

Elapsed time: 0.366019 seconds Total number of solutions: 276

Elapsed time: 0.344443 seconds Total number of solutions: 276

Elapsed time: 0.339823 seconds Total number of solutions: 276

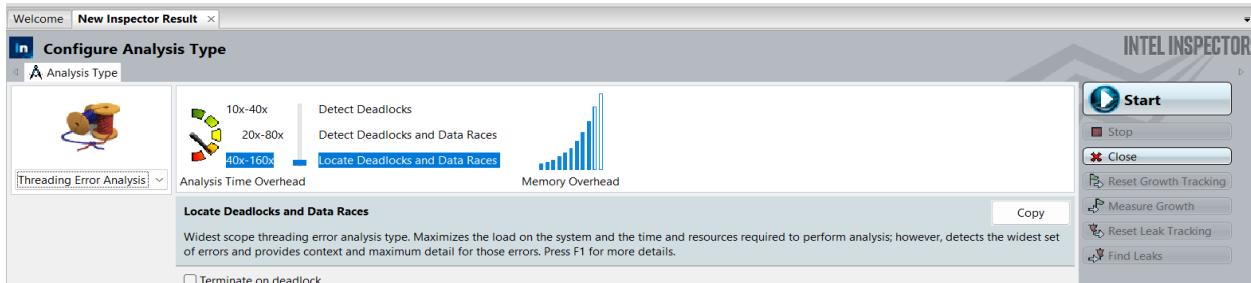
Elapsed time: 0.338272 seconds Total number of solutions: 276

پس در پایان این بخش، نسخه ای را که فقط موازی شده `kkn` اش سریال بهبود یافته است را نگه داشتیم. در فایل `AlirezaSerial.cpp` کد سریال بهبینه شده و در فایل `AlirezaParallel1.cpp` کد موازی اولیه و در فایل `AlirezaParallel2.cpp` کد موازی نهایی، موجود است.

بخش سوم، دیباگ:

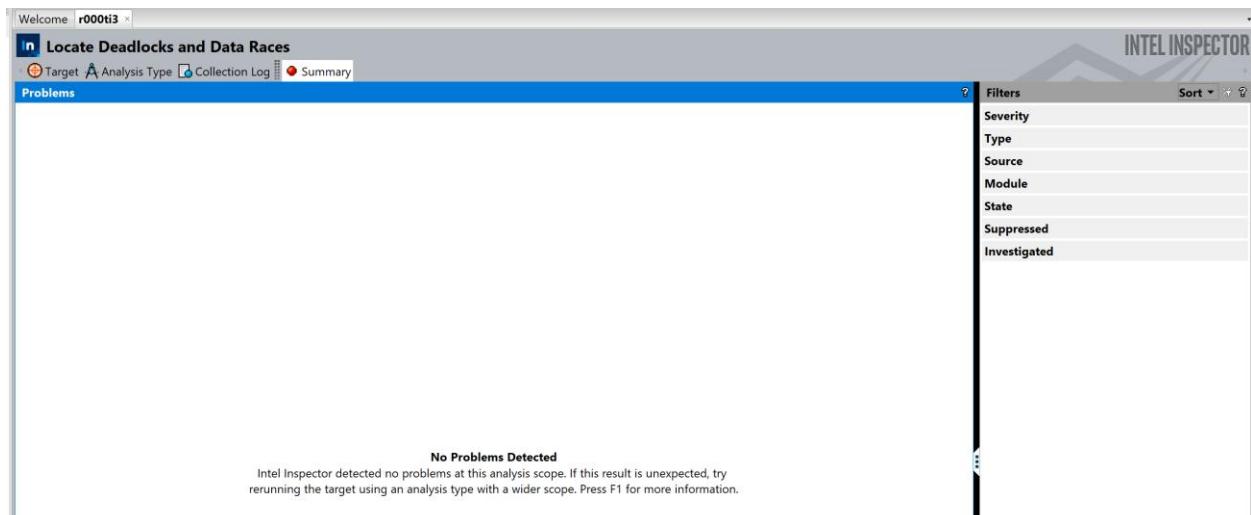
در این بخش به سراغ Intel Parallel Inspector رفتیم تا دللاک ها و یا دیتا ریس های احتمالی را ببینیم، ضمناً در تمامی بخش های قبلی خروجی برنامه که در واقع صفحه ای بود که اسب ها چیده شده بودند، با خروجی سریال اولیه توسط سایت: بررسی گردید و درست بودند صرفاً ترتیشان فرق داشت. <https://www.diffchecker.com/text-compare>

در Inspector این گزینه ها را انتخاب کردیم:



عکس 16: Inspector

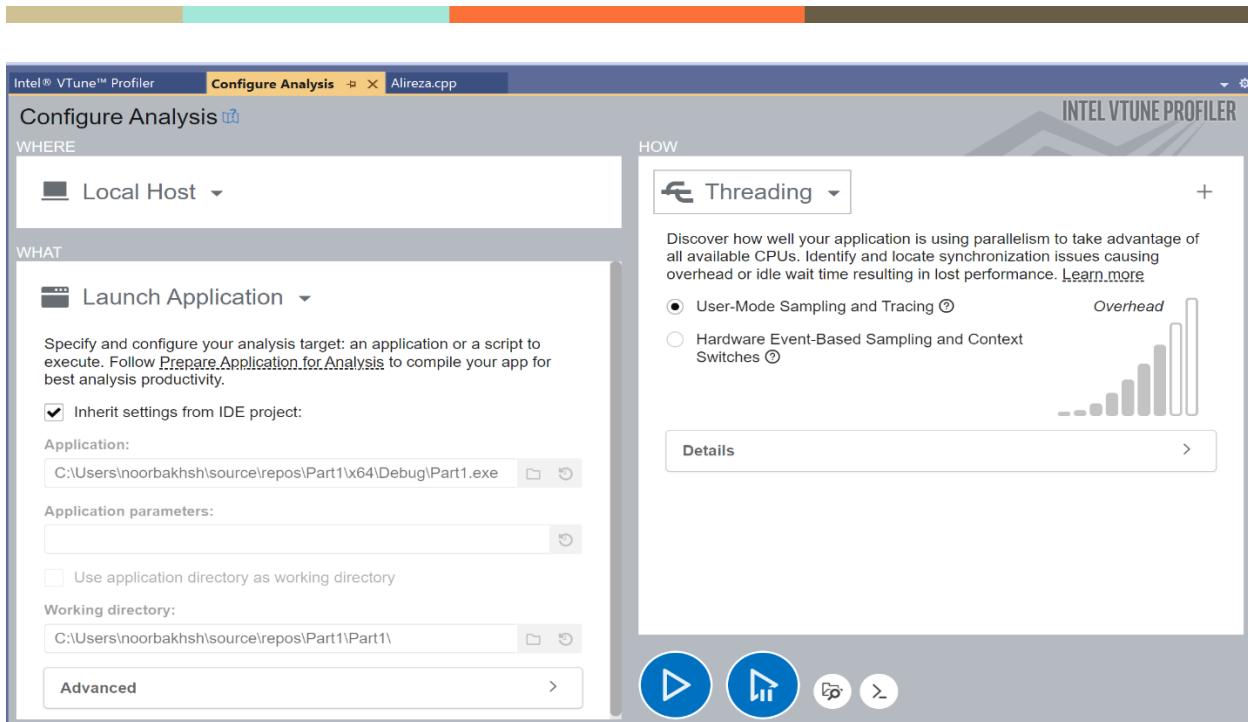
چون خروجی برنامه حین پیاده سازی اشتباه بود و سپس درست شد دیگر مشکلی در این بخش وجود نداشت و بیشتر با Inspector در بخش دوم این پروژه کار کردیم.



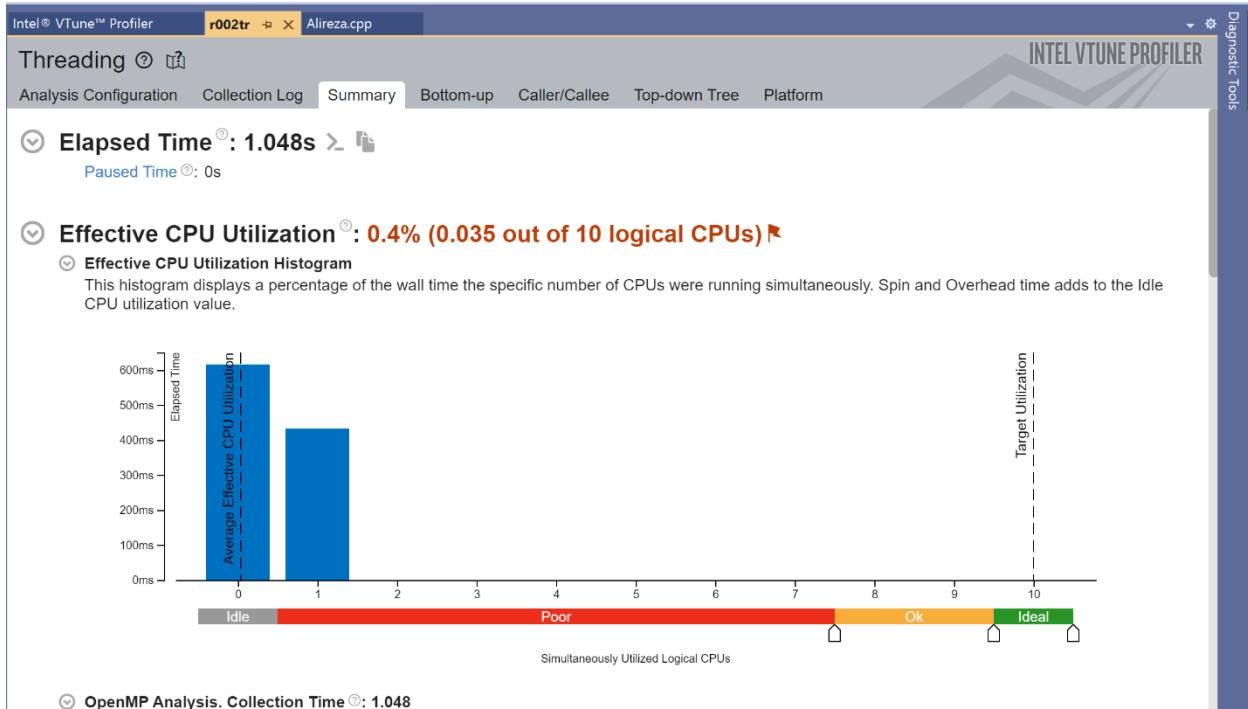
عکس 17: نتیجه تحلیل

بخش چهارم، بهینه سازی:

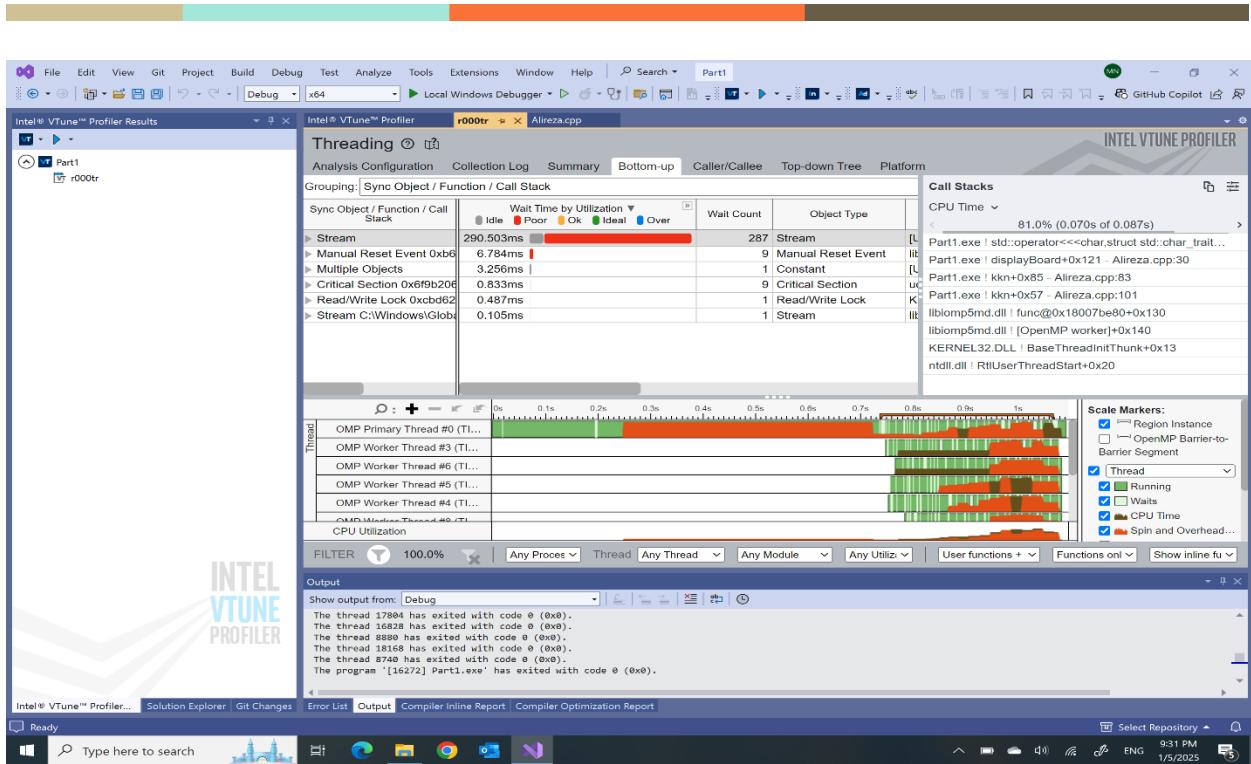
در ابتدای این بخش وقتی که بدنه موازی را در main داریم که می خواهیم kkn را برای بار اول صدا بزنیم، nowait را اضافه کردیم تا مابقی thread ها منتظر نباشند. سپس به سراغ VTune Profiler رفتیم تا چند گزارش بگیریم:



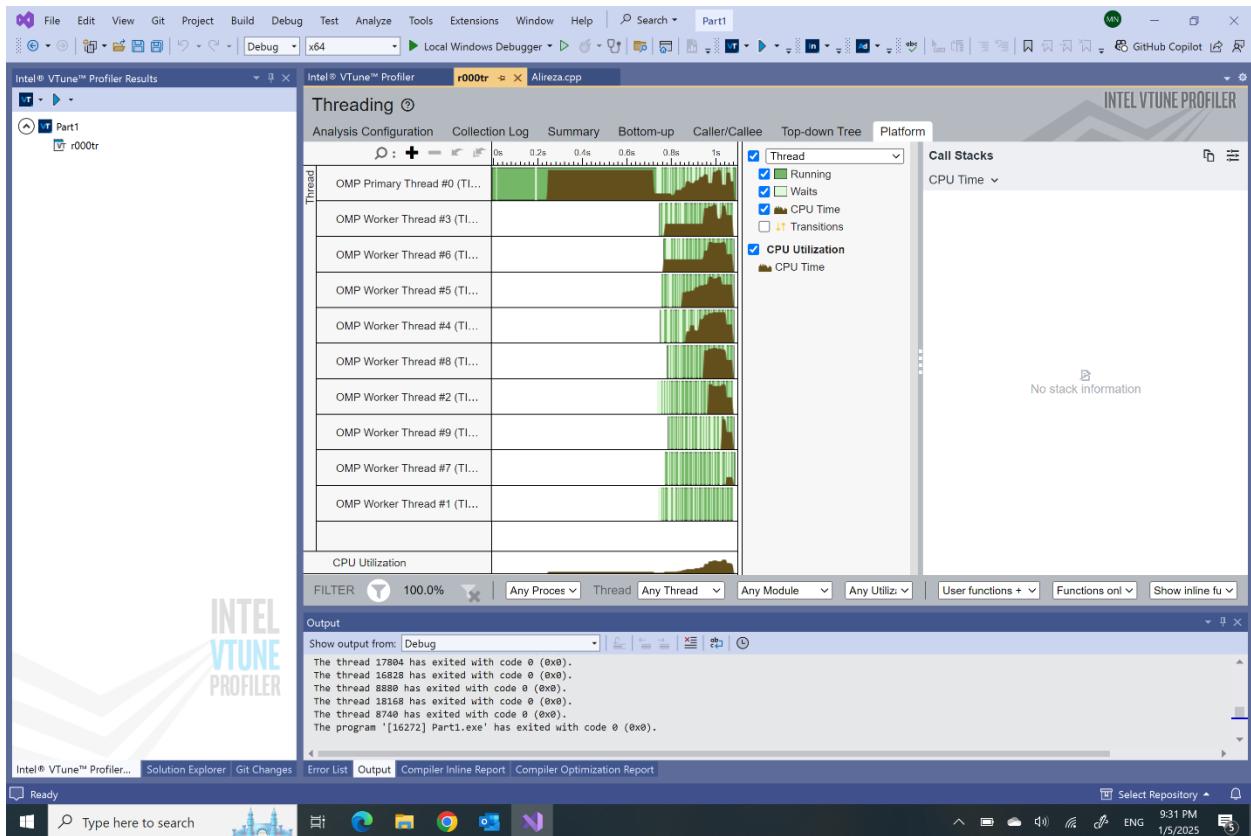
عکس ۱۸: VTune config



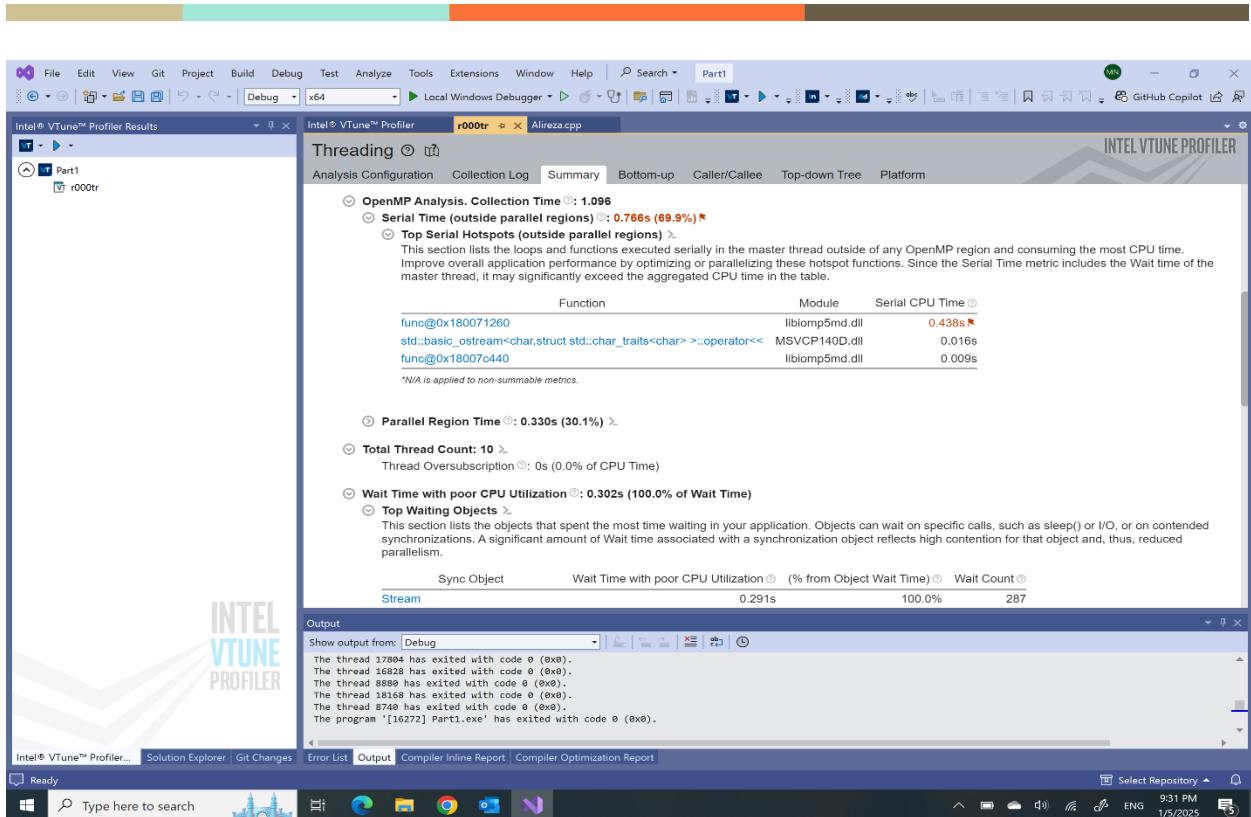
عکس ۱۹: خلاصه ای از گزارش



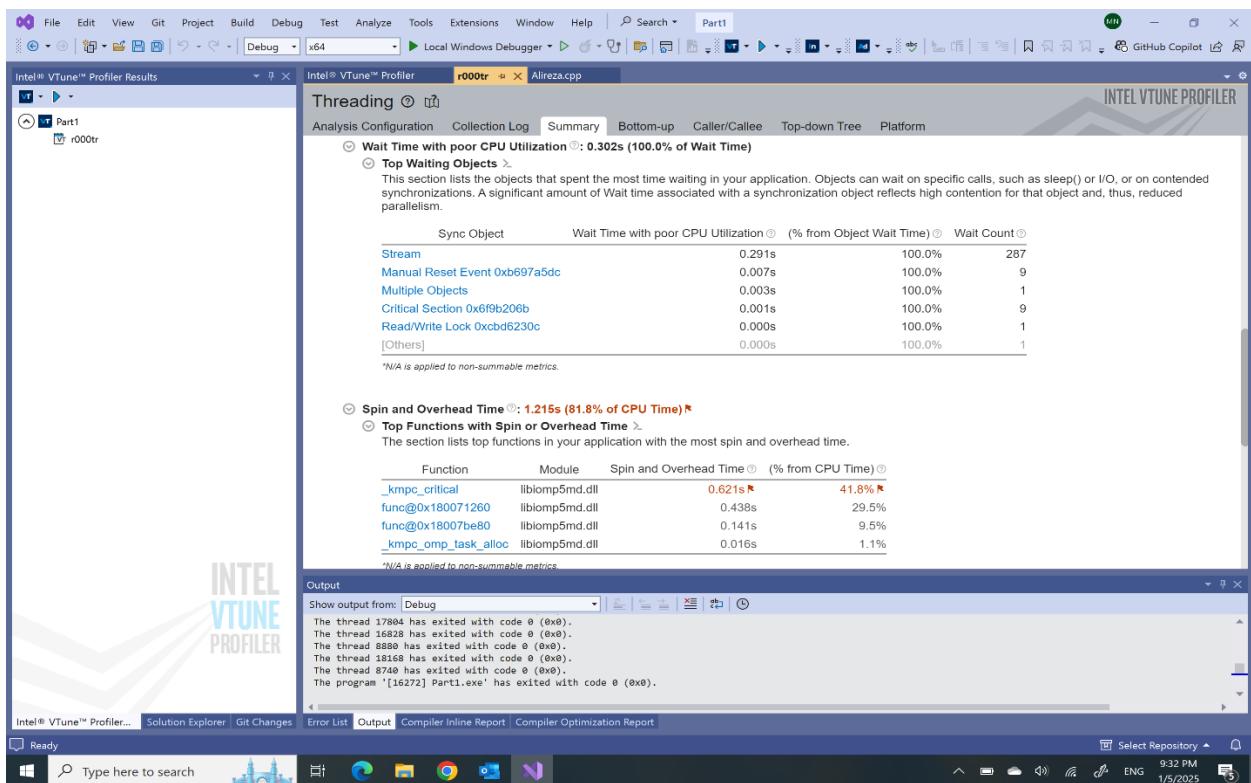
عکس 20: جزییات کارکرد هر ترد



عکس 21: ادامه جزییات

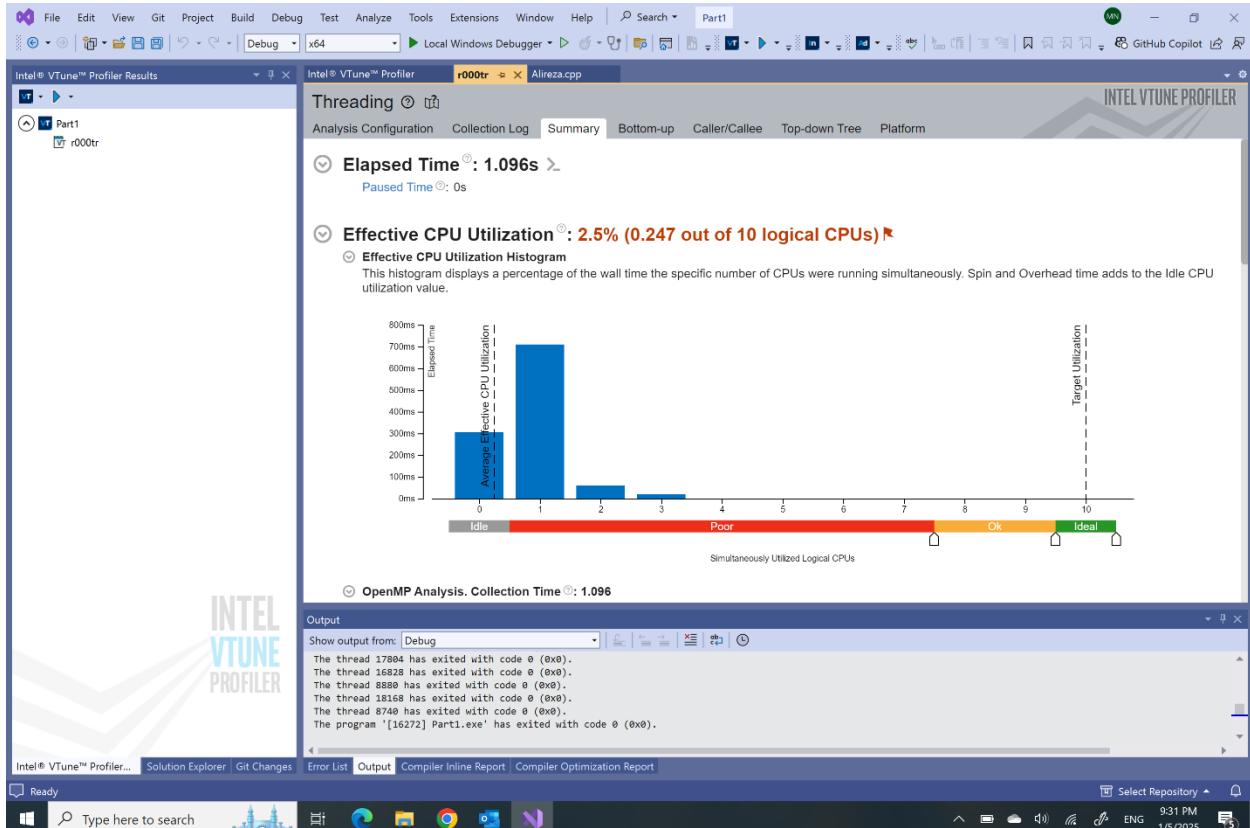


عکس 22



عکس 23

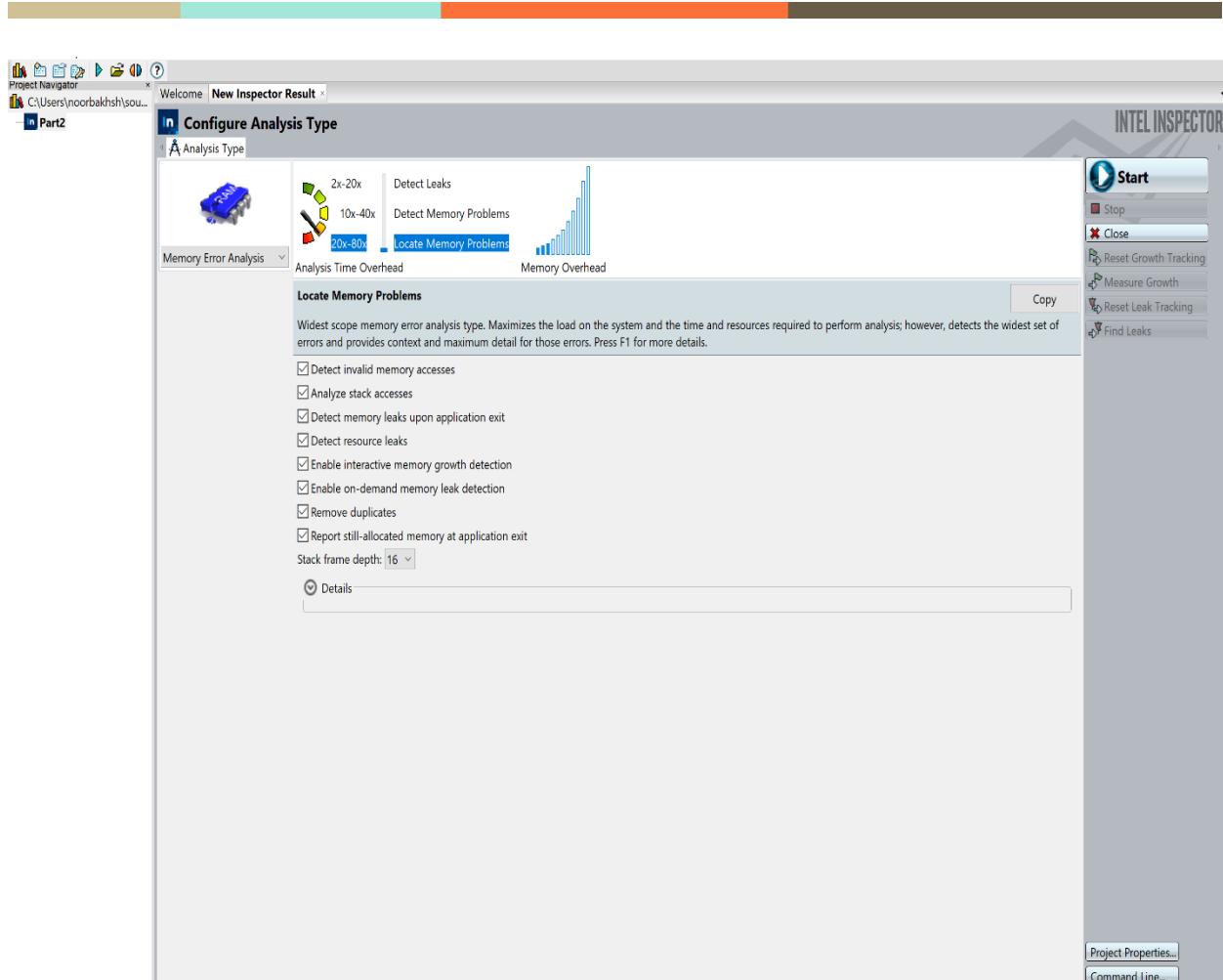
برای بهبود عملکرد کد، (kkn) در ابتدای تابع displayBoard(board) بود را به خارج بدنه critical استفاده کردیم. همچنین ما تا پایان این الگوریتم بازگشتی در حال ایجاد task هستیم که هزینه بر است پس تغییری در کد دادیم تا فقط برای زیر مساله های بزرگ task ایجاد شود.



عکس 24: گزارش بهبود یافته کد

بخش دوم(خواهرزاده همسر Danny)

با استفاده از Inspector این تحلیل را گرفتیم و در ادامه آن ها را رفع کردیم:



عکس 17 : Inspector

نتیجه اولیه آن این بود:

Intel Inspector - Locate Memory Problems

Problems

ID	Type	Sources	Modules	Object Size	State
P1	Memory leak	wife.c	part2.exe	6	New
P2	Invalid memory access	stdio.h; wife.c	part2.exe		New
P3	Invalid memory access	wife.c	part2.exe		New

Filters

- Severity: Error (3 item(s))
- Type: Invalid memory access (2 item(s)), Memory leak (1 item(s))
- Source: stdio.h (1 item(s)), wife.c (3 item(s))
- Module: part2.exe (3 item(s))
- State: New (3 item(s))
- Suppressed: Not suppressed (3 item(s))
- Investigated: Not investigated (3 item(s))

Code Locations: Memory leak

```

1 of 1 All
Description Source Function Module Object Size Offset Variable
Allocation site wife.c:65 initFirstMove part2.exe 6 block allocated at wife.c:65
63     "White's turn, enter the first move: ");
64     scanf("%d", &choice);
65     whiteToken = (char*) (malloc(sizeof(char) * MOVE_SIZE));
66     switch (choice)
67     {

```

Timeline

```

RtlActivateActivationContextUnsafeFast (14828)
RtlActivateActivationContextUnsafeFast (14828)

```

عکس 18: مشکلات اولیه

Intel Inspector - Memory leak

Allocation site - Thread RtlActivateActivationContextUnsafeFast (14828) (part2.exemain - wife.c:65)

```

wife.c Disassembly (part2.exe!0x163d)
44         currentToken[i] = previousToken[i] + 1;
45     }
46 }
47 }
48 return NULL;
49 }

50 char* initFirstMove(char* whiteToken)
51 {
52     if (strcmp(whiteToken, CHESS_TOKEN, strlen(CHESS_TOKEN)) != 0)
53         return whiteToken;
54
55     int choice;
56     printf(
57         "0: A King move\n"
58         "1: A Queen move\n"
59         "2: A Rook move\n"
60         "3: A Knight move\n"
61         "4: A Pawn move\n"
62         "White's turn, enter the first move: ");
63     scanf("%d", &choice);
64     whiteToken = (char*) (malloc(sizeof(char) * MOVE_SIZE));
65     switch (choice)
66     {
67     case(0):
68         strcpy(whiteToken, KING, strlen(KING));
69         break;
70     case(1):
71         strcpy(whiteToken, QUEEN, strlen(QUEEN));
72         break;
73     case(2):
74         strcpy(whiteToken, ROOK, strlen(ROOK));
75         break;
76     case(3):
77         strcpy(whiteToken, KNIGHT, strlen(KNIGHT));
78         break;
79     case(4):
80         strcpy(whiteToken, PAWN, strlen(PAWN));
81         break;
82     }
83     return whiteToken;
84 }

```

Call Stack

```

RtlActivateActivationContextUnsafeFast (14828)
RtlActivateActivationContextUnsafeFast (14828)
part2.exemain - wife.c:65
part2.exemain - wife.c:93
part2.exemain - exe_common.intl:78
part2.exemain - exe_common.intl:288
part2.exe!_scrt_common_main_seh() - exe_common.intl:330
part2.exe!_scrt_common_main() - exe_common.intl:330
part2.exe!mainCRTStartup - exe_main.cpp:16
kernel32.dll!BaseThreadInitThunk
ntdll.dll!RtlUserThreadStart

```

عکس 19: جایی که مشکل memory leak رخ می دهد

The screenshot shows the Intel Inspector interface with the title bar "Welcome | r001mi3". The main window displays an "Invalid memory access" report for thread "RtActivateActivationContextUnsafeFast (10992) (part2.exe!main - wife.c:92)". The left pane shows the source code for "wife.c" with line 92 highlighted. The right pane shows the call stack starting from "part2.exe!main - wife.c:92" up to "ntdll.dll!RtlUserThreadStart".

```

1 Welcome | r001mi3
2 [In] Invalid memory access
3 Target Analysis Type Collection Log Summary Sources
4 Read - Thread RtActivateActivationContextUnsafeFast (10992) (part2.exe!main - wife.c:92)
5 wife.c Disassembly (part2.exe!0x1769)
6
7 58     "0: A King move\n"
8 59     "1: A Queen move\n"
9 60     "2: A Rook move\n"
10 61     "3: A Knight move\n"
11 62     "4: A Pawn move\n"
12 63     "White's turn, enter the first move: "
13 64 scanf("%d", &choice);
14 whiteToken = (char*)malloc(sizeof(char) * MOVE_SIZE);
15 switch (choice)
16 {
17 case(0):
18     strcpy(whiteToken, KING, strlen(KING));
19     break;
20 case(1):
21     strcpy(whiteToken, QUEEN, strlen(QUEEN));
22     break;
23 case(2):
24     strcpy(whiteToken, ROOK, strlen(ROOK));
25     break;
26 case(3):
27     strcpy(whiteToken, KNIGHT, strlen(KNIGHT));
28     break;
29 case(4):
30     strcpy(whiteToken, PAWN, strlen(PAWN));
31     break;
32 }
33 return whiteToken;
34
35 int main(int argc, char* argv[])
36 {
37     char* token = createBlackToken(argv[1]);
38     printf("Token: %s\n", token);
39     free(token);
40     token = createWhiteToken(token);
41     printf("White's move: %s\n", initFirstMove(token));
42     free(token);
43     return EXIT_SUCCESS;
44 }
45

```

عکس 20: ناشی از آن leak در این خط از کد هم به مشکل می خوریم چون توکنی نیست و داریم آن را پاس می دهیم

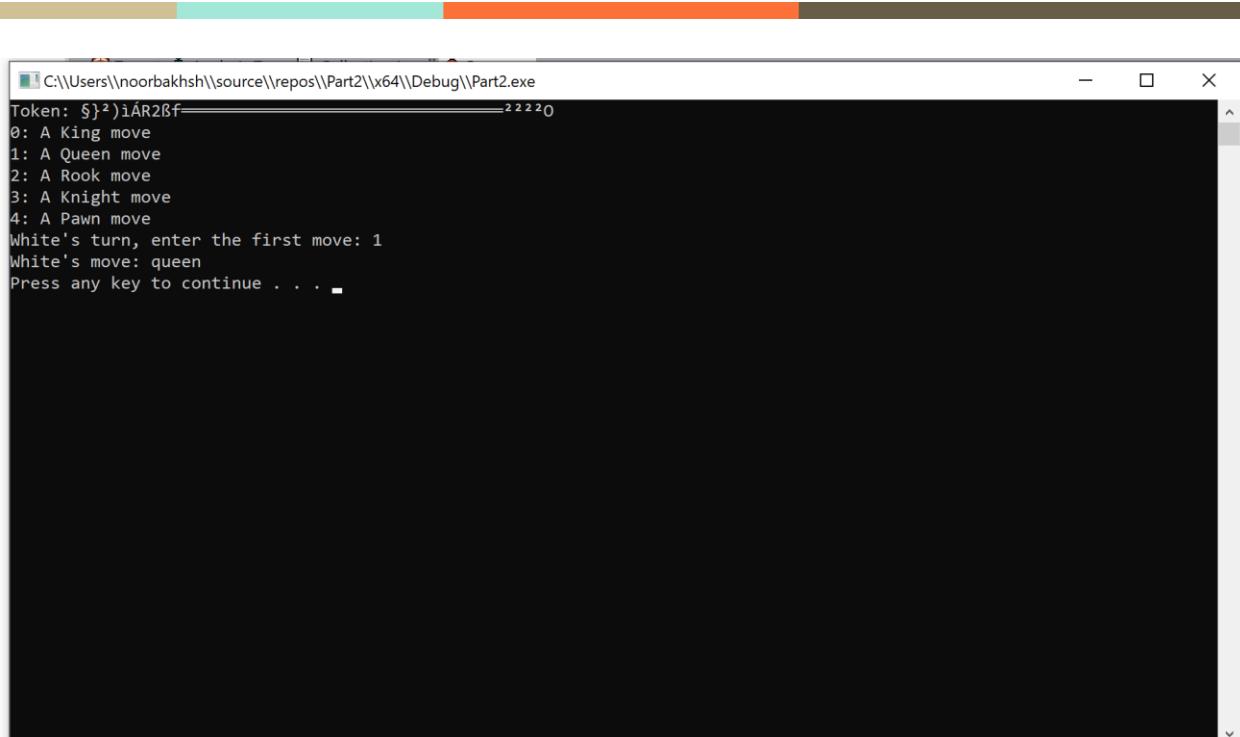
The screenshot shows the Intel Inspector interface with the title bar "Welcome | r001mi3". The main window displays an "Uninitialized memory access" report for thread "RtActivateActivationContextUnsafeFast (10992) (part2.exe!main - wife.c:93)". The left pane shows the source code for "wife.c" with line 93 highlighted. The right pane shows the call stack starting from "part2.exe!main - wife.c:93" up to "part2.exe!mainCRTStartup".

```

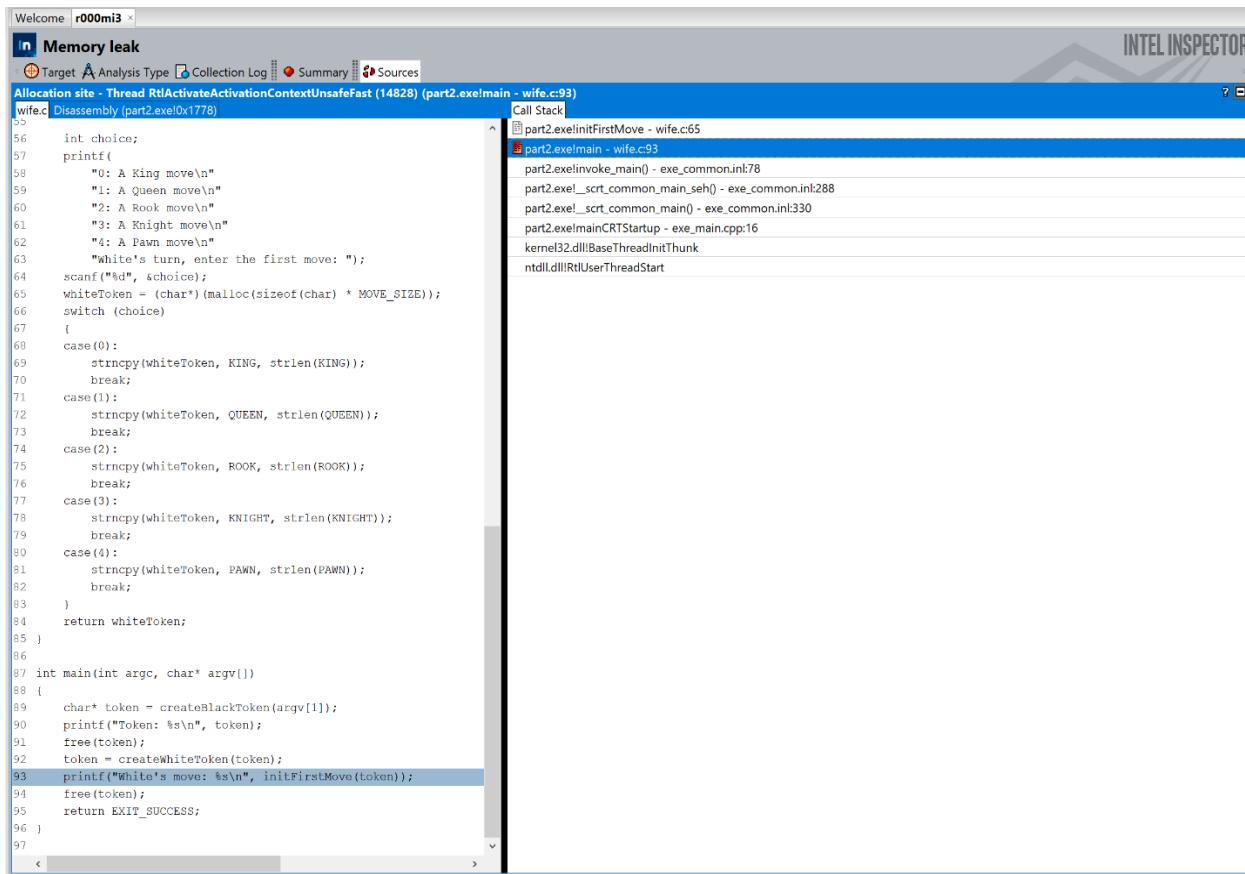
1 Welcome | r001mi3
2 [In] Uninitialized memory access
3 Target Analysis Type Collection Log Summary Sources
4 Read - Thread RtActivateActivationContextUnsafeFast (10992) (part2.exe!main - wife.c:93)
5 wife.c Disassembly (part2.exe!0x1787)
6
7 57     printf(
8 58         "0: A King move\n"
9 59         "1: A Queen move\n"
10 60         "2: A Rook move\n"
11 61         "3: A Knight move\n"
12 62         "4: A Pawn move\n"
13 63         "White's turn, enter the first move: "
14 64 scanf("%d", &choice);
15 whiteToken = (char*)malloc(sizeof(char) * MOVE_SIZE);
16 switch (choice)
17 {
18 case(0):
19     strcpy(whiteToken, KING, strlen(KING));
20     break;
21 case(1):
22     strcpy(whiteToken, QUEEN, strlen(QUEEN));
23     break;
24 case(2):
25     strcpy(whiteToken, ROOK, strlen(ROOK));
26     break;
27 case(3):
28     strcpy(whiteToken, KNIGHT, strlen(KNIGHT));
29     break;
30 case(4):
31     strcpy(whiteToken, PAWN, strlen(PAWN));
32     break;
33 }
34 return whiteToken;
35
36 int main(int argc, char* argv[])
37 {
38     char* token = createBlackToken(argv[1]);
39     printf("Token: %s\n", token);
40     free(token);
41     token = createWhiteToken(token);
42     printf("White's move: %s\n", initFirstMove(token));
43     free(token);
44     return EXIT_SUCCESS;
45 }
46

```

عکس 21: وقتی که ورودی نادرستی را به برنامه می دهیم به این مساله می خوریم



عکس 22: نمونه ای از خروجی برنامه



عکس 23: مساله ای که اینجا هست به خاطر اینکه توکن در خط بالایی به درستی مقدار نگرفته است

```

Welcome r000m3 x
In Invalid memory access
Target Analysis Type Collection Log Summary Sources
Read - Thread RtlActivateActivationContextUnsafeFast (14828) (part2.exe!createWhiteToken - wife.c:44)
wife.c Disassembly [part2.exe!0x1591]
24     tokenHolder[i] = rand() % 255;
25 }
26     return tokenHolder;
27 }
28     return NULL;
29 }
30
31 char* createWhiteToken(char* previousToken)
32 {
33     char* currentToken = (char*) (malloc(sizeof(char)) * TOKEN_SIZE);
34     int i = 0;
35     if (previousToken != NULL)
36     {
37         while (1)
38         {
39             if (i >= TOKEN_SIZE)
40                 return currentToken;
41             else if (i < strlen(CHESS_TOKEN))
42                 currentToken[i] = CHESS_TOKEN[i];
43             else
44                 currentToken[i] = previousToken[i] + 1;
45             i++;
46         }
47     }
48     return NULL;
49 }
50
51 char* initFirstMove(char* whiteToken)
52 {
53     if (strcmp(whiteToken, CHESS_TOKEN, strlen(CHESS_TOKEN)) != 0)
54         return whiteToken;
55
56     int choice;
57     printf(
58         "0: A King move\n"
59         "1: A Queen move\n"
60         "2: A Rook move\n"
61         "3: A Knight move\n"
62         "4: A Pawn move\n"
63         "White's turn, enter the first move: ");
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96

```

عکس 24: چون انداز بافر مناسب نیست، به آدرس نادرستی از آن دسترسی پیدا می‌کنیم

```

Welcome r001m3 x
In Invalid memory access
Target Analysis Type Collection Log Summary Sources
Read - Thread RtlActivateActivationContextUnsafeFast (10992) (part2.exe!main - wife.c:90)
wife.c Disassembly [part2.exe!0x1754]
57     printf(
58         "0: A King move\n"
59         "1: A Queen move\n"
60         "2: A Rook move\n"
61         "3: A Knight move\n"
62         "4: A Pawn move\n"
63         "White's turn, enter the first move: ");
64     scanf("%d", &choice);
65     whiteToken = (char*) (malloc(sizeof(char)) * MOVE_SIZE);
66     switch (choice)
67     {
68     case(0):
69         strcpy(whiteToken, KING, strlen(KING));
70         break;
71     case(1):
72         strcpy(whiteToken, QUEEN, strlen(QUEEN));
73         break;
74     case(2):
75         strcpy(whiteToken, ROOK, strlen(ROOK));
76         break;
77     case(3):
78         strcpy(whiteToken, KNIGHT, strlen(KNIGHT));
79         break;
80     case(4):
81         strcpy(whiteToken, PAWN, strlen(PAWN));
82         break;
83     }
84     return whiteToken;
85 }
86
87 int main(int argc, char* argv[])
88 {
89     char* token = createBlackToken(argv[1]);
90     printf("token: %s\n", token);
91     free(token);
92     token = createWhiteToken(token);
93     printf("White's move: %s\n", initFirstMove(token));
94     free(token);
95     return EXIT_SUCCESS;
96 }

```

عکس 25: وقتی که توکنی سمت نمی‌کنیم، NULL می‌رود به تابع درست کردن توکن سیاه و NULL بر می‌گردد

در وله اول، مشکلات ناشی از این که یک حافظه پویا درست تخصیص نشود را چاپ کردیم. این مساله را برای نتامی جاهایی که تخصیص حافظه پویا داریم، انجام دادیم. ضمناً تابع `createBlackToken` مساله ای داشت که اگر `argv[1]` خالی بود، در خود `createBlackToken` بر می گشت و چاپ می شد و سپس `free` می شد که غلط است و اصلاح شد که اگر خالی بود، در خود حافظه اش خالی شود.

```
char* tokenHolder = (char*)(malloc(sizeof(char) * TOKEN_SIZE));
if (tokenHolder == NULL)
{
    perror("Memory allocation failed");
    exit(EXIT_FAILURE);
}
```

وقتی که داخل یک حلقه، عملیات seeding random number generator را برای `iteration` داریم، باعث می شود که در هر `strand(time(0))` ریست شود و مقادیر متفاوتی را داشته باشیم، پس برای اینکه برنامه ما به درستی تصادفی باشد، `createWhiteToken` را به خارج حلقه بردهیم. ضمناً برای وقتی که `name` خالی است، حالتی در نظر گرفتیم که `free` شود (در هم این کار را کردیم):

```
if (name != NULL)
{
    srand((unsigned int)time(0));
    for (int i = 0; i < TOKEN_SIZE; ++i)
    {
        tokenHolder[i] = rand() % 255;
    }
}
else
{
    free(tokenHolder);
    return NULL;
}

return tokenHolder;
```

همچنین حلقه بی نهایتی هم که در `createWhiteToken` وجود داشت را با `for` پیاده سازی کردیم که همان `logic` را دارد:

```
if (previousToken != NULL)
{
    for (int i = 0; i < TOKEN_SIZE; ++i)
    {
        if (i < strlen(CHESS_TOKEN))
            currentToken[i] = CHESS_TOKEN[i];
        else
            currentToken[i] = previousToken[i] + 1;
    }
}
else
{
    free(currentToken);
    return NULL;
}

return currentToken;
```

در تابع `main` داشتیم: `token = createWhiteToken(token)` `free(token)` که باعث می شد که صحیح شد.

وقتی که می خواهیم از کاربر ورودی بگیریم، چک می کنیم که ورودی که وارد می کند معقول باشد:

```
if (scanf("%d", &choice) != 1)
{
    fprintf(stderr, "Invalid input\n");
    exit(EXIT_FAILURE);
}
```

ای که پیش تر داشتیم، حالت default را نداشت که برای آن اضافه کردیم که کاربر به جز گزینه هایی که به آن داده ایم وارد نکند (همانند این مساله را در اسلاید شماره 16 درس که بحث static security مطرح شده بود، داشتیم):

```
default:
{
    fprintf(stderr, "Invalid move\n");
    free(moveToken);
    return NULL;
}
```

در آخر این تابع هم مطمئن می شویم که رشته با null خاتمه یابد که جلو گیری کنیم از buffer overflow و در آن 7 افزایش می دهیم تا knight را هم بتواند در خودش جا دهد:

```
moveToken[MOVE_SIZE - 1] = '\0';
```

در تابع main چک می کنیم که اگر توکن سیاه ساخته شده خالی نیست، ابتدا آن را چاپ کنیم و سپس free کنیم.

```
if (token != NULL)
{
    printf("Token: %s\n", token);
    free(token);
}
```

بعد از تمامی این تغییرات باز هم مشکل Invalid memory access را داشتیم که فهمیدیم به خاطر سایز توکن هایی است که داریم malloc می کنیم که چون در تمامی برنامه به صورت خاتمه با NULL بودند آن را 1 واحد افزایش دادیم تا کاراکتر آخر که NULL است را هم شامل شود که پس از این تغییر دیگر مشکل حافظه ای در کد وجود نداشت.

نمونه ای تخصیص جدید:

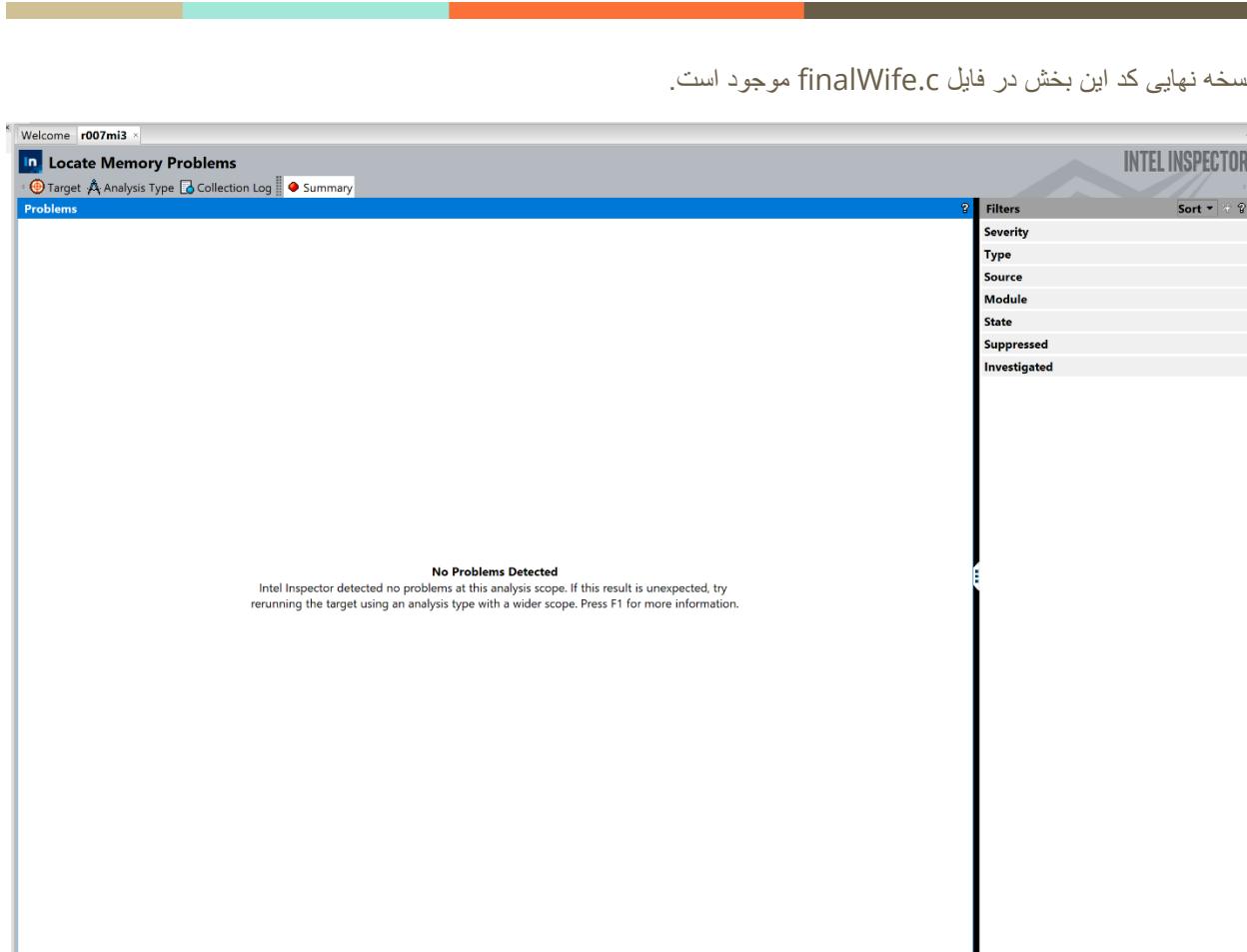
```
char* tokenHolder = (char*)(malloc(sizeof(char) * (TOKEN_SIZE + 1)));
```

نمونه ای از انتهای توکن های جدید:

```
tokenHolder[TOKEN_SIZE] = '\0';
```

برای برطرف کردن مشکلی هم که در خط: (token = createWhiteToken(token)) وجود داشت، یک توکن موقت گرفتیم و توکن را در آن کپی کردیم:

```
char* tempToken = (char*)malloc(sizeof(char) * (TOKEN_SIZE + 1));
if (tempToken == NULL)
{
    fprintf(stderr, "Failed to dup token\n");
    free(token);
    return EXIT_FAILURE;
}
memcpy(tempToken, token, TOKEN_SIZE + 1);
```



عکس 26: گزارش نهایی کد درست شده



در آخر این بخش هم باید بگوییم که نمره قابل قبولی را از نظر ما کسب نکرد!