



درس برنامه نویسی چند هسته ای

محاسبه دترمینان





❖ مشخصات سخت افزار سخت افزار

مشخصات سخت افزاری که این برنامه در آن اجرا شده برابر است با:

1. مدل پردازنده مورد استفاده

Processor					
Name	Intel Core i7 7500U				
Code Name	Kaby Lake-U/Y	Max TDP	15.0 W		
Package	Socket 1515 FCBGA				
Technology	14 nm	Core VID	1.075 V		
Specification	Intel® Core™ i7-7500U CPU @ 2.70GHz				
Family	6	Model	E	Stepping	9
Ext. Family	6	Ext. Model	8E	Revision	B0
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3				



CPU																													
	<table><tr><td colspan="2">Intel Core i7-7500U</td><td colspan="2">14 nm</td></tr><tr><td>Stepping</td><td>H0</td><td>Cores/Threads</td><td>2 / 4</td></tr><tr><td>Codename</td><td colspan="2">Kaby Lake-U</td><td>μCU</td><td>8E</td></tr><tr><td>SSPEC</td><td colspan="2">SR2ZV, SR341</td><td colspan="2">Prod. Unit</td></tr><tr><td>Platform</td><td colspan="4">BGA1356/BGA1515</td></tr><tr><td>Cache</td><td colspan="4">2x32 + 2x32 + 2x256 + 4M</td></tr></table>	Intel Core i7-7500U		14 nm		Stepping	H0	Cores/Threads	2 / 4	Codename	Kaby Lake-U		μCU	8E	SSPEC	SR2ZV, SR341		Prod. Unit		Platform	BGA1356/BGA1515				Cache	2x32 + 2x32 + 2x256 + 4M			
Intel Core i7-7500U		14 nm																											
Stepping	H0	Cores/Threads	2 / 4																										
Codename	Kaby Lake-U		μCU	8E																									
SSPEC	SR2ZV, SR341		Prod. Unit																										
Platform	BGA1356/BGA1515																												
Cache	2x32 + 2x32 + 2x256 + 4M																												
CPU #0	▼																												
TDP	15 W																												

2. تعداد هسته و نخ های CPU و بیشینه فرکانس کاری آن.

➤ Number Of Processor Packages (Physical):	1
➤ Number Of Processor Cores:	2
➤ Number Of Logical Processors:	4

⊖ CPU MFM (LowPower):	400.0 MHz = 4 x 100.0 MHz
⊖ CPU LFM (Minimum):	400.0 MHz = 4 x 100.0 MHz
⊖ CPU HFM (Base):	2900.0 MHz = 29 x 100.0 MHz
⊖ CPU Turbo Max:	3500.0 MHz = 35 x 100.0 MHz [Unlocked]
⊖ Turbo Ratio Limits:	35x (1-4c)
⊖ CPU Current:	2694.7 MHz = 27 x 99.8 MHz @ 0.9019 V
⊖ LLC/Ring Maximum:	3300.0 MHz = 33.00 x 100.0 MHz
⊖ LLC/Ring Current:	2495.1 MHz = 25.00 x 99.8 MHz
⊖ System Agent Current:	798.4 MHz = 8.00 x 99.8 MHz

3. تعداد سطوح حافظه ی نهان و اندازه ی هر کدام. همچنین اندازه ی خطوط حافظه نهان.

Cache and TLB

L1 Cache:	Instruction: 2 x 32 KBytes, Data: 2 x 32 KBytes
L2 Cache:	Integrated: 2 x 256 KBytes
L3 Cache:	4 MBytes
Instruction TLB:	2MB/4MB Pages, Fully associative, 8 entries
Data TLB:	4 KB Pages, 4-way set associative, 64 entries

4. حجم حافظه ی اصلی سیستم.

General information

Total Memory Size:	16 GBytes
--------------------	-----------

Current Performance Settings

Maximum Supported Memory Clock:	1066.7 MHz
Current Memory Clock:	1064.6 MHz
Current Timing (tCAS-tRCD-tRP-tRAS):	15-15-15-35
Memory Channels Supported:	2
Memory Channels Active:	1

(منابع:

1. برنامه CPU-Z
2. برنامه HWiNFOPortable

5. کامپایلر مورد استفاده:

Visual C++

❖ الگوریتم محاسبه دترمینان و نحوه پیاده سازی

در این برنامه دترمینان یک ماتریس به کمک روش LU محاسبه شده است؛ به این صورت که ابتدا ماتریس L و U محاسبه می شود، همانطور که مشخص است دترمینان ماتریس ورودی برابر حاصل ضرب دترمینان ماتریس L و U می باشد، و دترمینان ماتریس های L و U نیز به دلیل اینکه ماتریس های مثلثی هستند برابر حاصل ضرب درایه های قطری ماتریس می باشد؛ در این روش درایه های قطری ماتریس L برابر 1 می باشد به همین دلیل دترمینان ماتریس اصلی برابر دترمینان ماتریس U می باشد.

با توجه به اینکه در صورت داشتن ماتریس U دترمینان ماتریس اصلی به سادگی محاسبه می گردد؛ ابتدا باید روشی مناسب برای پیدا کردن ماتریس های L و U پیدا کرد. در این برنامه از روش Block LU decomposition استفاده شده است، در این روش ابتدا اندازه بلوک ماتریس داخلی انتخاب می گردد.

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 \\ L_{10} & L_{11} \end{bmatrix} * \begin{bmatrix} U_{00} & U_{01} \\ 0 & U_{11} \end{bmatrix}$$

همانطور که در شکل بالا مشاهده می کنید؛ یک ماتریس مربعی به اندازه b وجود دارد، با این کار ماتریس به 4 قسمت متفاوت تقسیم می شود، بنابراین هر دفعه یه ماتریس مربعی به با اندازه ای وابسته به اندازه ماتریس اصلی انتخاب می شود و سپس به کمک محاسبات زیر بخش های مختلف ماتریس تقسیم شده محاسبه می گردد:

$$\begin{cases} 1 & A_{00}=L_{00}U_{00} \\ 2 & A_{10}=L_{10}U_{00} \\ 3 & A_{01}=L_{00}U_{01} \\ 4 & A_{11}=L_{10}U_{01}+L_{11}U_{11} \end{cases}$$

ابتدا با توجه به فرمول 1 ماتریس U_{00} و L_{00} به کمک روش حذف گوسی (Gaussian elimination) محاسبه می گردد. سپس به کمک فرمول 2 ماتریس L_{10} و به کمک فرمول 3 ماتریس U_{01} محاسبه می گردد؛ سپس به کمک فرمول 4 ماتریس A_{11} محاسبه می گردد، تمامی مراحل بالا دوباره بر روی ماتریس A_{11} انجام می شود تا این ماتریس به اندازه کافی کوچک شود و پس از اینکه به اندازه کافی کوچک شد به کمک روش حذف گوسی آخرین بلوک ماتریس محاسبه می شود.

در این روش برای محاسبه ماتریس های L_{10} و U_{01} نیاز است تا ابتدا ماتریس های U_{00} و L_{00} محاسبه شوند به همین دلیل ابتدا مراحل محاسبه این ماتریس ها موازی شده اند، پس از محاسبه این ماتریس ها ماتریس های L_{10} و U_{01} محاسبه می شوند، این ماتریس ها می توانند به صورت مستقل از یک دیگر محاسبه شوند به همین دلیل به کمک دستور sections این ماتریس ها به صورت موازی با یک دیگر محاسبه می شوند؛ پس از آن ماتریس A_{11} باید محاسبه شود که به دلیل نیاز به داده های مرحله های قبل پس از اتمام آن ها محاسبه می شود محاسبه این ماتریس نیز به صورت مستقل موازی شده است، پس از محاسبه این ماتریس دوباره مراحل بالا بر روی این ماتریس اجرا می شود.

(منابع: <https://lon91ong.github.io/LU-Decomposition/> و

[https://www.youtube.com/watch?v=mQyZ3yLk_RY&list=LL5bc9wmYTjnNzHHTWdaFfCQ&index=](https://www.youtube.com/watch?v=mQyZ3yLk_RY&list=LL5bc9wmYTjnNzHHTWdaFfCQ&index=7&t=0s)

[7&t=0s](https://www.youtube.com/watch?v=mQyZ3yLk_RY&list=LL5bc9wmYTjnNzHHTWdaFfCQ&index=7&t=0s) و

[https://www.youtube.com/watch?v=E8aBJsC0bY8&list=LL5bc9wmYTjnNzHHTWdaFfCQ&index=](https://www.youtube.com/watch?v=E8aBJsC0bY8&list=LL5bc9wmYTjnNzHHTWdaFfCQ&index=8&t=0s)

[8&t=0s](https://www.youtube.com/watch?v=E8aBJsC0bY8&list=LL5bc9wmYTjnNzHHTWdaFfCQ&index=8&t=0s)

❖ نحوه خواندن فایل و محاسبه میانگین زمان محاسبات و روال اجرای برنامه

در این برنامه هر فایل txt شامل یک ماتریس می باشد که هر سطر فایل برابر سطر ماتریس می باشد (فایل های ورودی همانند test case های داده شده می باشد)، شکل زیر یک ماتریس مربعی 4 در 4 را نمایش می دهد:

```
1.0037537766655475 1.1785154361362633 1.8659215077441278 1.7169222594241342
0.5680074465163122 3.3576580853412139 5.1284469082654764 4.4803094978285394
0.4838984344004639 0.9483822311549621 3.0549487241214468 2.9271181101774335
0.8167699209570605 1.7172482192162857 3.3632841906784439 6.5198228312556132
```

این برنامه ابتدا تمام فایل های موجود در یکی از پوشه های `data_in` را یکی یکی می خواند و با محاسبه `LU` برای هر یک از آن ها دترمینان آن ماتریس را محاسبه می کند و سپس مقدار دترمینان محاسبه شده را در یک فایل مشترک در پوشه `data_out` می نویسد، در پوشه `data_out` فایل ها با فرمت `N_results` ذخیره شده اند که `N` برابر ابعاد ماتریسی که دترمینان آن محاسبه شده می باشد (مقدار دترمینان ماتریس های با بعد یکسان در یک فایل نوشته شده است)، در پوشه `data_in` ماتریس ها بر اساس ابعادشان در پوشه های متفاوت قرار گرفته اند؛ این پوشه در کنار فایل برنامه اصلی قرار گرفته است.

در خط 156 می توان آدرس فایل ورودی و در خط 160 می توان آدرس خروجی برنامه را براساس آدرس های جدید در صورت نیاز تغییر داد.

```
150 int main()
151 {
152     double average_all_elapsedtime = 0;
153     int num_of_files = 0;
154     // open a file in write mode.
155     ofstream outfile;
156     outfile.open("data_out\\512_results.txt");
157     omp_set_num_threads(4);
158
159     namespace fs = std::experimental::filesystem; //std::filesystem
160     std::string path = "data_in\\512";
```

همچنین سائز ماتریس های ورودی در خط 11 توسط مقدار `N` و تکرار محاسبه های هر فایل برای میانگین گیری زمان اجرای در خط 12 توسط مقدار `epoch` مشخص می گردد، تغییر مقدار `N` بر اساس ابعاد فایل های ورودی ضروری می باشد.

```
10 // Dimension of input square matrix  
11 #define N 512  
12 #define epoch 10
```

روال اجرای این برنامه به این صورت است که ابتدا یک ماتریس از پوشه انتخاب شده خوانده می شود، سپس برای محاسبه میانگین زمان انجام شدن محاسبه ها بر روی فایل خوانده شده 10 بار، که این مقدار می تواند توسط مقدار *epoch* کم یا زیاد شود، محاسبه ها تکرار می شود و زمان اجرا در هر بار و میانگین زمان اجرا 10 بار اندازه گیری می شود، این کار برای تمامی ماتریس های موجود در پوشه انجام می شود و در نهایت مقدار میانگین این زمان های اجرا محاسبه ها برای ماتریس ها اندازه گیری شده و به عنوان میانگین زمان اجرای ماتریس های به ابعاد مشخص شده در نظر گرفته می شود.

برای مثال اگر در پوشه ماتریس های به ابعاد 4، 10 فایل وجود داشته باشد هر کدام از این فایل ها 10 بار اجرا می شوند و در نهایت میانگین زمان اجرای محاسبه های این 10 فایل به عنوان میانگین زمان اجرای محاسبه های ماتریس با آن ابعاد در نظر گرفته می شود (برابر با 100 بار اجرای ماتریس های به ابعاد 4).

در این محاسبه ها زمان خواندن ماتریس از فایل و نوشتن دترمینان ماتریس در فایل اندازه گیری نشده است.

❖ موازی سازی و Fine Tuning

ابتدا برای موازی سازی این برنامه در نظر گرفته می شود که کدام قسمت از مرحله های این برنامه می تواند به صورت هم زمان با یک دیگر اجرا شود؛ سپس به موازی سازی قسمت های مختلف کد بر اساس نیاز پرداخته می شود.

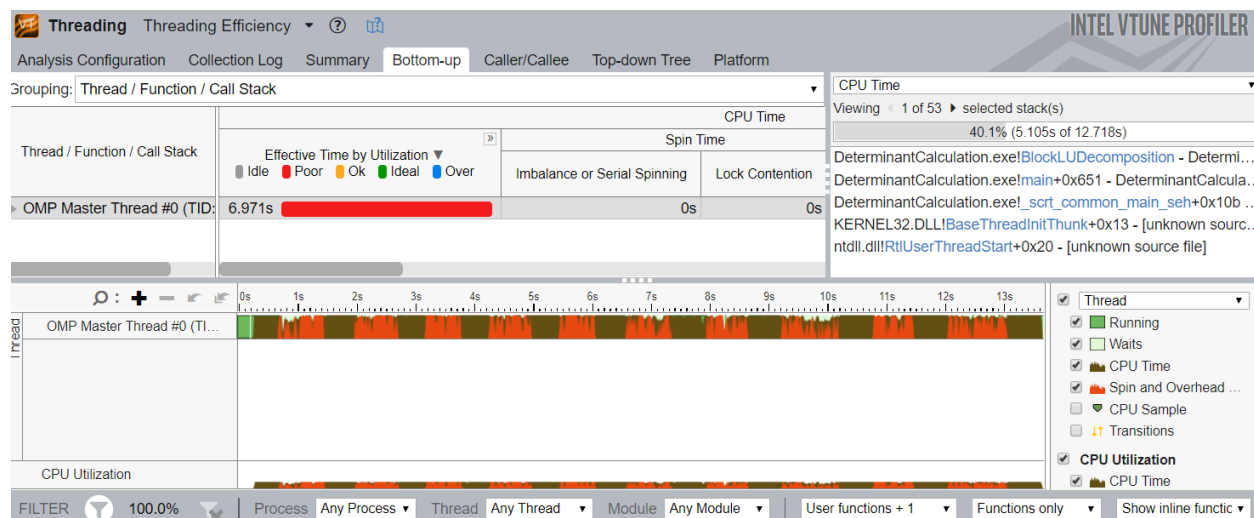
همانطور که در بخش الگوریتم محاسبه دترمینان و نحوه پیاده سازی گفته شده، تنها محاسبه ماتریس های L_{10} و U_{01} و این می تواند به صورت هم زمان با یک دیگر انجام شود. این موازی سازی به کمک *sections* انجام شده است بدین صورت که 2 نخ مسئول محاسبه ماتریس اول و دو نخ دیگر نیز مسئول محاسبه ماتریس دوم می شوند.

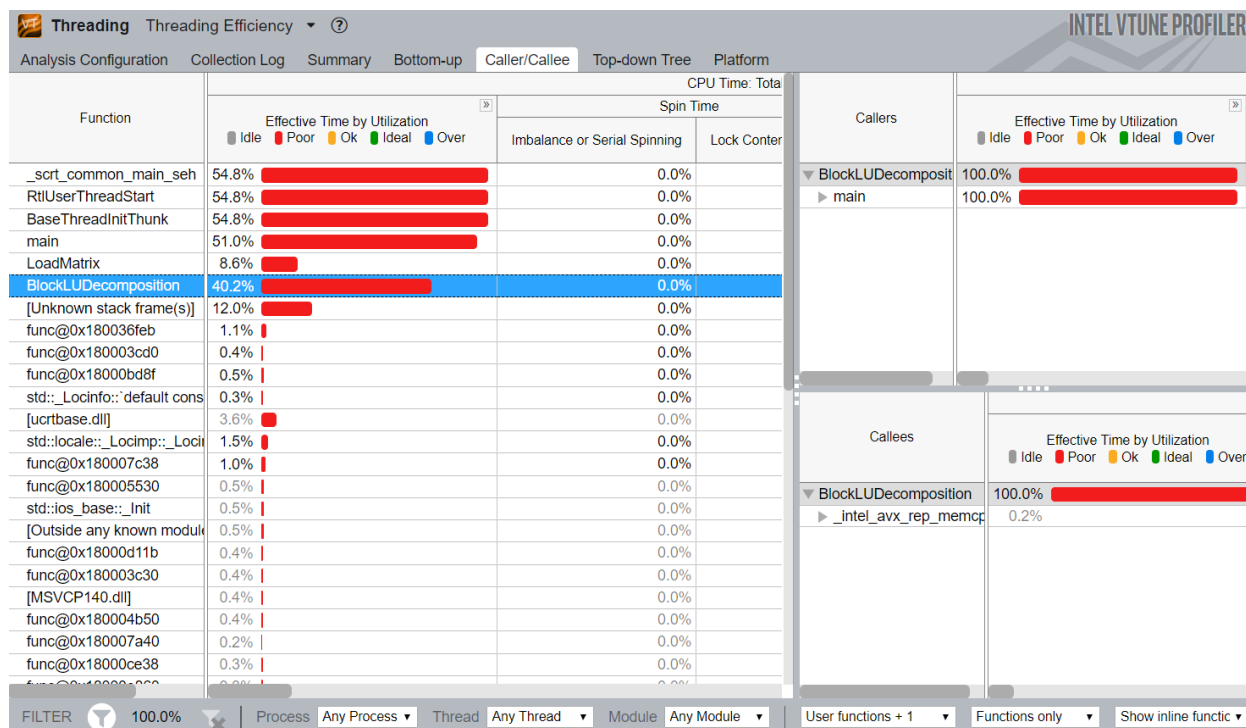
برای جزئیات بیشتر ابتدا برنامه سری پروفایل می گردد، که نتیجه آن برابر است با:

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time ②
BlockLUdecomposition	DeterminantCalculation.exe	5.105s
func@0x180036feb	MSVCP140.dll	3.554s 🚩
func@0x180003cd0	MSVCP140.dll	1.255s 🚩
func@0x18000bd8f	MSVCP140.dll	0.612s
std::_Locinfo::`default constructor closure'	MSVCP140.dll	0.461s
[Others]		1.732s





همانطور که در شکل های بالا مشاهده می گردد، تابع *BlockLUdecomposition* و *LoadMatrix* سهم قابل توجهی از اجرا برنامه را دارند، تابع *LoadMatrix* امکان موازی سازی ندارد چرا که به کمک آن یک ماتریس از فایل خوانده می شود و در صورت موازی سازی ترتیب خواندن از فایل به هم خورده و در نتیجه ماتریس ورودی اشتباه خوانده می شود و نتیجه نیز دچار مشکل می شود؛ اما تابع *BlockLUdecomposition* که به کمک آن در هر مرحله بخش هایی از ماتریس های L و U محاسبه می شود دارای بخش هایی است که می تواند به صورت موازی با هم اجرا شود. با توجه به مطالب عنوان شده تنها محاسبه دو ماتریس L_{10} و U_{01} به صورت همزمان با یک دیگر انجام می شود و در بخش های دیگر صرفا انجام خود محاسبه ها موازی شده است.

به طور کلی قسمت های زیر در بخش محاسبه های برنامه موازی شده است:

- تابع *GaussianElimination* در تابع *BlockLUdecomposition* برای محاسبه ماتریس های U_{00} و L_{00} و زمانی که ماتریس به اندازه کافی کوچک شده باشد فراخوانی می شود، این تابع دارای دو حلقه تو در تو می باشد که در حلقه اول ماتریس U_{00} مقدار دهی اولیه می شود و سپس در حلقه بعدی سطرهای ماتریس U_{00} و ستون های ماتریس L_{00}

محاسبه می شود، این مقدار ها از طریق مقدار سطر های قبلی ماتریس U_{00} محاسبه می شود، بنابراین نمی توان حلقه اول این ماتریس را به صورت موازی انجام داد، بنابراین در این روش تکرار های حلقه تو در تو اول در سطح اول و تکرار های حلقه تو در تو بعدی در سطح دوم به صورت موازی اجرا می شوند.

- دو ماتریس L_{10} و U_{01} به کمک sections موازی با یک دیگر اجرا می شوند، بدین صورت که تکرار های حلقه سطح اول هر یک توسط دو نخ به صورت موازی اجرا می شود.
 - برای محاسبه A_{11} نیاز به انجام یک ضرب ماتریسی می باشد، تکرار های حلقه سطح دوم این ضرب نیز به صورت موازی اجرا می شود.
 - در نهایت نیز برای محاسبه دترمینان نیاز است تا قطر های ماتریس U در یک دیگر ضرب شوند که این تکرار های حلقه ضرب نیز به صورت موازی اجرا می شوند و در انتها حاصل نخ ها به کمک reduction با یک دیگر ادغام می شود.
- سپس صحت خروجی برنامه برای ماتریس های $4 * 4$ بررسی می گردد، شکل زیر سمت چپ برابر جواب های مورد انتظار و شکل سمت راست برابر نتیجه های بدست آمده توسط برنامه می باشد:

4_answers.txt - Notepad

File Edit Format View Help

```
14.6200895201910033
5.7604192742161322
15.9928300588481385
68.8765293588872538
12.8152178525630571
54.6318177353355594
22.9703417031200452
22.9712972463682235
11.1530945938064008
32.1508372277252263
```

4_results.txt - Notepad

File Edit Format View Help

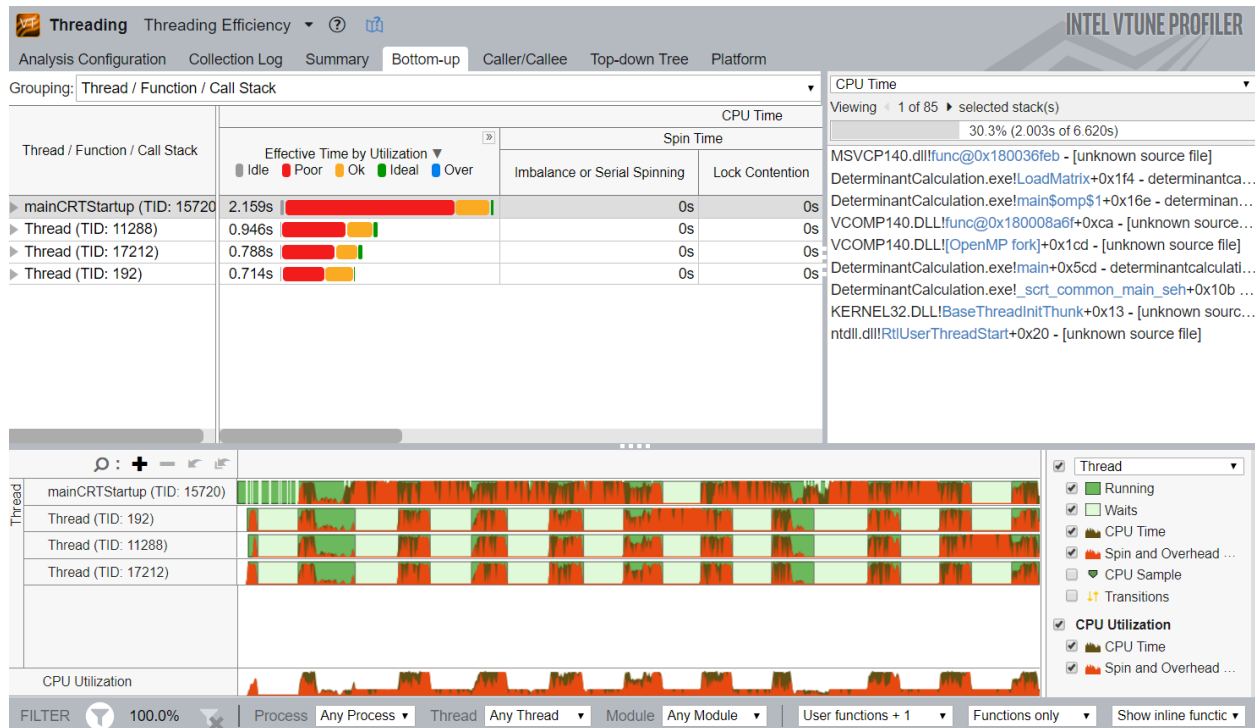
```
14.6201
5.76042
15.9928
68.8765
12.8152
54.6318
22.9703
22.9713
11.1531
32.1508
```

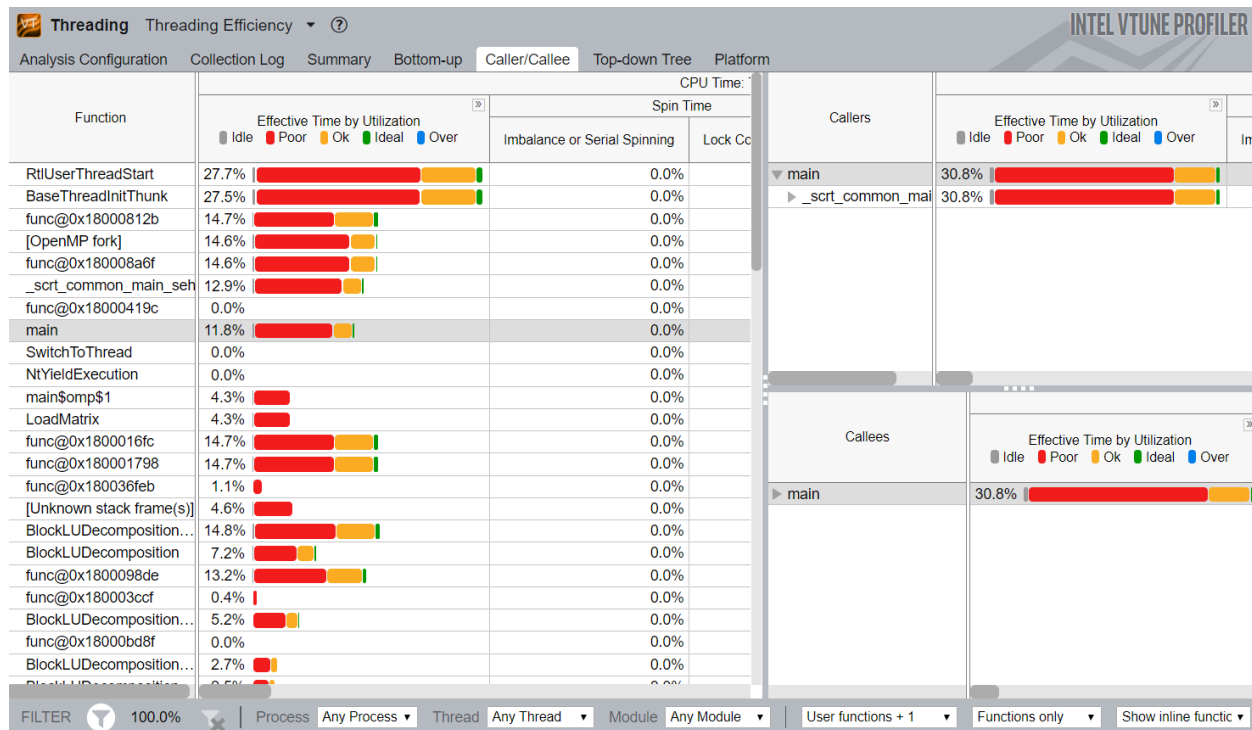
حال که صحت درستی خروجی برنامه تعیین شد، برنامه موازی پروفایل می گردد که نتایج آن برابر است با:

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time ②
NtYieldExecution	ntdll.dll	6.111s 🚩
func@0x180036feb	MSVCP140.dll	2.594s 🚩
BlockLUDecomposition\$omp\$4	DeterminantCalculation.exe	2.468s
func@0x180008a6f	VCOMP140.DLL	1.277s 🚩
func@0x180003ccf	MSVCP140.dll	0.895s
[Others]		3.277s 🚩





همانطور که در شکل های بالا مشاهده می شود، تابع *BlockLUDecomposition* توسط نخ های مختلف اجرا شده و زمان اجرای آن نیز نسبت به حالت سریال کاهش قابل توجهی یافته است که باعث سریع تر شدن اجرای برنامه شده است.

این برنامه با 4 نخ و برای ماتریس های به ابعاد 512 اجرا شده است؛ برای *Fine Tuning* زمان اجرای محاسبه ها برای ماتریس های 512 و تعداد نخ های متفاوت اندازه گیری می گردد:

میانگین زمان اجرای محاسبه ها (Sec)	تعداد نخ های هر بخش <i>section</i>	تعداد نخ های بخش <i>sections</i>	تعداد نخ های کل برنامه
0.042444	2	4	4
0.138173	4	8	4
0.044781	4	8	8
0.065737	8	16	16

همانطور که مشاهده می کنید به دلیل اینکه این برنامه دارای محاسبه های زیادی می باشد و *computed bound* است با افزایش تعداد نخ ها میانگین زمان اجرا محاسبه ها افزایش می یابد، همچنین بیشتر کردن تعداد نخ های *sections* باعث افزایش شدید میانگین زمان اجرا می شود؛ بنابراین

این برنامه با 4 نخ که برابر تعداد نخ هایی می باشد که این دستگاه به صورت موازی می تواند اجرا کند، کمترین زمان اجرا محاسبه ها را بدست می آورد، چرا که این یک برنامه *computed bound* می باشد و با بیشتر کردن نخ ها سربار *context switch* باعث افزایش میانگین زمان اجرای محاسبه ها می شود.

❖ محاسبه تسریع

در این بخش تسریع ماتریس های با ابعاد مختلف محاسبه می گردد؛ برای این کار ابتدا زمان اجرای سری این برنامه به ازای ابعاد مختلف زیر اندازه گیری می گردد، که برابر است با:

ابعاد ماتریس	میانگین زمان اجرای محاسبه ها (Sec)
4	$10^{-6} <$
8	0.000001
16	0.000002
32	0.000017
64	0.000210
256	0.011373
512	0.086476

حال میانگین زمان اجرای محاسبه ها در روش موازی با 4 نخ فعال اندازه گیری می شود و سپس تسریع محاسبه می گردد:

ابعاد ماتریس	میانگین زمان اجرای محاسبه ها (Sec)	تسریع
4	0.000029	$Speed-up = \frac{nearly\ 0}{0.000029} * 100 \approx 0\%$
8	0.000034	$Speed-up = \frac{0.000001}{0.000034} * 100 \approx 2.94\%$
16	0.000104	$Speed-up = \frac{0.000002}{0.000104} * 100 \approx 1.92\%$
32	0.000973	$Speed-up = \frac{0.000017}{0.000973} * 100 \approx 1.74\%$
64	0.006250	$Speed-up = \frac{0.000210}{0.006250} * 100 \approx 3.36\%$
256	0.009547	$Speed-up = \frac{0.011373}{0.009547} * 100 \approx 119.12\%$
512	0.041539	$Speed-up = \frac{0.086476}{0.041539} * 100 \approx 208.18\%$

همانطور که در جدول بالا مشاهده می شود، سریار موازی سازی برای ماتریس های کوچک بسیار زیاد می باشد و تسریع در آن ها بسیار کم است و با موازی اجرا کردن برنامه نه تنها سرعت برنامه بیشتر نشده بلکه سرعت اجرای محاسبه ها به دلیل سریار های موازی سازی کاهش نیز می یابد، اما با بزرگ کردن سائز ماتریس تسریع برنامه افزایش می یابد؛ برای مثال با افزایش سائز ماتریس از $64 * 64$ به $256 * 256$ تسریع از 3.36% به 119.12% افزایش می یابد و این مقدار برای ماتریس $512 * 512$ نیز افزایش می یابد.

(در پوشه قرار داده شده به همراه گزارش، فایل سریال برنامه به نام *DeterminantCalculation_serial.cpp* و فایل موازی شده برنامه به نام *DeterminantCalculation_parallel.cpp* قرار داده شده است.)