

به نام خدا

گزارش آزمایش ۷

محمد مهدی نظری ۹۹۳۱۰۶۱ - علی نوروزیگی ۹۹۳۱۰۶۲

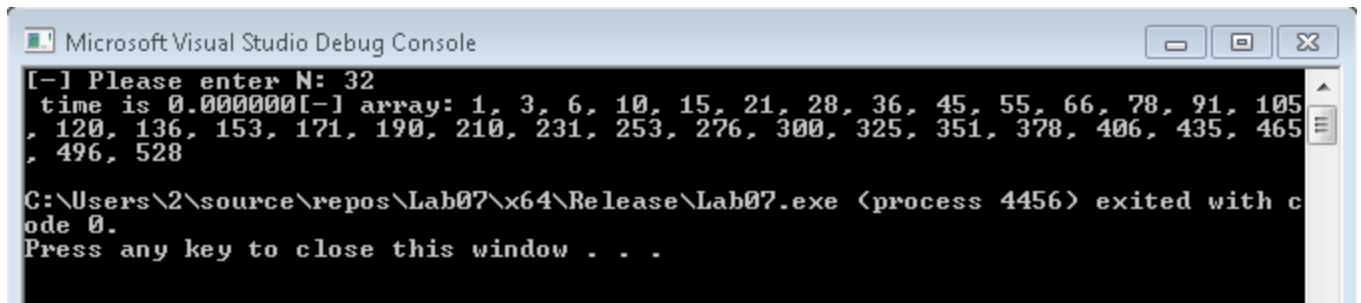
گام اول :

در پیاده سازی این الگوریتم از ۲ تابع کرنل استفاده شده که اولی مراحل ۱ و ۲ در تصویر را انجام داده و کرنل دوم مرحله ۳ را بر روی خروجی کرنل اول انجام داده و خروجی نهایی را تولید میکند. (آرایه c . آرایه a آرایه ورودی و آرایه d آرایه میانی در مرحله ۲ شکل است)

همچنین در اجرای الگوریتم از یک ثابت به نام NUM_TASK_PER_THREAD استفاده شده که تعیین میکند که هر ترد خروجی چند خانه را تولید کند یا اینکه آرایه به چند قسمت تقسیم شود (تعداد کل بخش ها $n/\text{NUM_TASK_PER_THREAD}$ میشود)

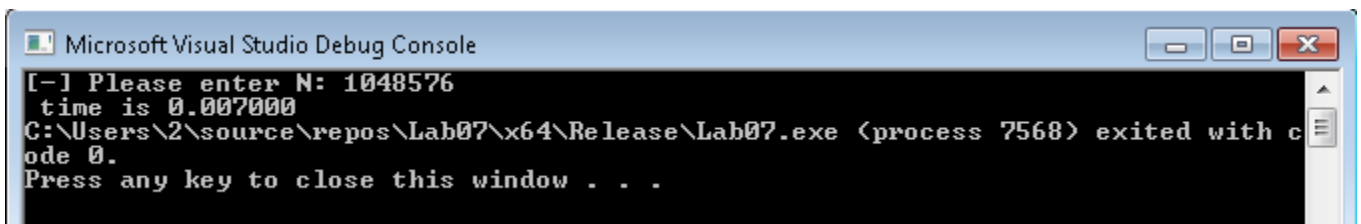
برای تست هم فرض شده که یک بلوک ۱۰۲۴ عددی از تردها داریم.

نتیجه صحت الگوریتم به ازای آرایه ۳۲ عددی در gpu :



```
Microsoft Visual Studio Debug Console
[-] Please enter N: 32
time is 0.000000[-] array: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351, 378, 406, 435, 465, 496, 528
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 4456) exited with code 0.
Press any key to close this window . . .
```

نتیجه زمان به ازای آرایه ۱۰۴۸۵۷۶ (۱۰۲۴ * ۱۰۲۴) عددی در gpu :



```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
time is 0.007000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 7568) exited with code 0.
Press any key to close this window . . .
```

نتیجه زمان به ازای آرایه ۱۰۴۸۵۷۶ (۱۰۲۴ * ۱۰۲۴) عددی در cpu :

```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
Elapsed time in sec 0.018000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 7940) exited with c
ode 0.
Press any key to close this window . . .
```

SpeedUp حالت موازی در gpu نسبت به cpu :

$$0.018 / 0.007 = 2.57$$

۱.

با توجه به مقایسه زمان اجرا و تسريع متوجه ميشويم كه استفاده از gpu بهتر است (با توجه به تعداد بالای نخ در gpu اين نتیجه هم قابل انتظار بود)

۲.

تا حد مناسبی باعث افزایش سرعت میشود اما از یک حدی بیشتر باعث کندتر شدن میشود . باید حد مناسب را محاسبات و آزمایش بدست آورد.

* در اجرای بخش shared memory به ارور زیر برخوردیم که متاسفانه برطرف نشد :

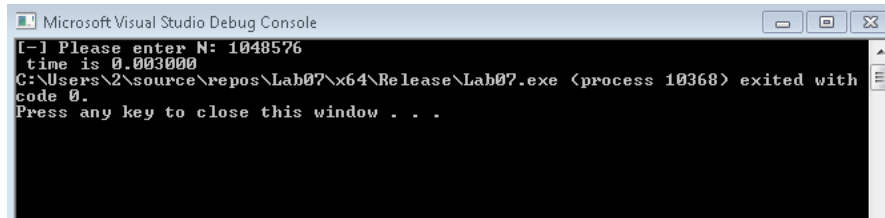
C4477	printf formatting %f requires an argument of type 'double', but variable argument 1 has type 'int'	Lab07	kernel.cu	344
LNK2001	unresolved external symbol "enum cudaError __cdecl prefixWithCuda1(int *,int const *,unsigned int,int *)" (?prefixWithCuda1@@YA?AW4cudaError@@PEAHPEBHIO@Z)	Lab07	kernel.cu.obj	1
LNK1120	1 unresolved externals	Lab07	Lab07.exe	1

گام دوم:

در این گام الگوریتم Hillis and Steele را روی GPU پیاده می کنیم. کد پیاده سازی شده به پیوست موجود است (البته در کد برای حالت float تغییر داده شده اما برای حالت int نیز درست است). برای اطمینان از درستی الگوریتم ابتدا با تعداد ورودی ۳۲ برنامه را بررسی می کنیم:

```
Microsoft Visual Studio Debug Console
[-] Please enter N: 32
time is 0.000000[-] array: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105,
, 120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351, 378, 406, 435, 465
, 496, 528
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 4796) exited with c
ode 0.
Press any key to close this window . . .
```

سپس برنامه را مشابه گام ۱ با اندازه ورودی ۱۰۴۸۵۷۶ بر روی یک بلوک با تعداد ۱۰۲۴ نخ نیز بررسی می کنیم:



```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
time is 0.003000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 10368) exited with
code 0.
Press any key to close this window . . .
```

با مقایسه عدد بدست آمده در این گام با عدد بدست آمده در گام اول، تسریع را محاسبه می کنیم:

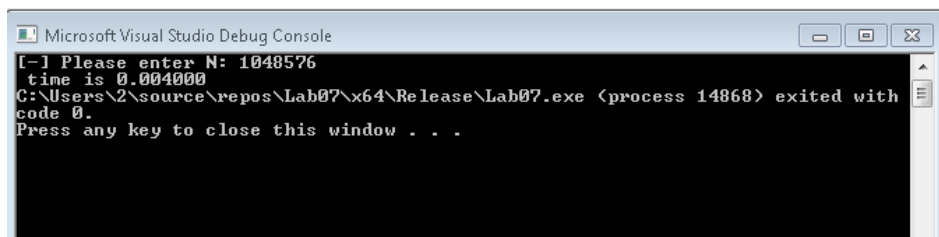
$$\text{Speed up} = 0.007 / 0.003 = 2.33$$

می بینیم که در صورتی که الگوریتم Hillis and Steele روی GPU پیاده شود، مقدار تسریع زیادی نسبت به حالتی که روی CPU پیاده شود به ما می دهد. همچنین نسبت به پیاده سازی استاتیک نیز سریعتر است. برای مقایسه با حالت cpu، در آزمایش ۴ به ازای ورودی ۱۰۰۰۰۰ به زمان ۰,۰۰۲۲۲۱ رسیدیم که اگر آن را بصورت خطی به ورودی ۱۰۴۸۵۷۶ تقریب بزنیم به زمان ۰,۰۲۳۲ میرسیم که تسریع تقریبی برابر است با :

$$0.0232 / 0.003 = 7.762$$

اکنون برنامه را با تعداد بلوک و اندازه های متفاوت اجرا می کنیم.

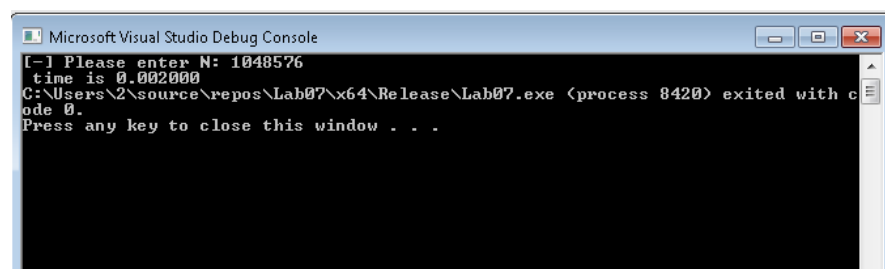
ابتدا ۱۰۴۸۵۷۶ را با یک بلوک ۵۱۲ تایی اجرا می کند تا تاثیر کوچکتر شدن تعداد نخها با ثابت گرفتن بلوک را بررسی کنیم:



```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
time is 0.004000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 14868) exited with
code 0.
Press any key to close this window . . .
```

می بینیم که زمان نسبت به انجام برنامه با ۱۰۲۴ نخ افزایش داشته است.

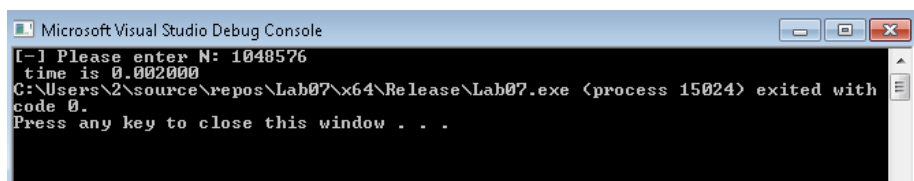
در ادامه، برنامه با ورودی ۱۰۴۸۵۷۶ را با ۲ بلوک ۱۰۲۴ تایی بررسی می کنیم تا تاثیر افزایش بلوک ها مشخص شود:



```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
time is 0.002000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 8420) exited with c
ode 0.
Press any key to close this window . . .
```

می بینیم که همانطور که انتظار داریم، با افزایش تعداد بلوک، زمان اجرای برنامه سریعتر از حالت اولیه شد.

در ادامه نیز ۱۰۴۸۵۷۶ ورودی را با تعداد ۱۶ بلوک ۱۰۲۴ تایی اجرا می کنیم. نتیجه:



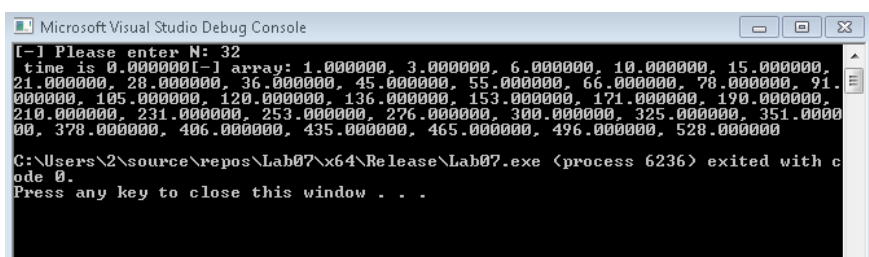
```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
time is 0.002000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 15024) exited with code 0.
Press any key to close this window . . .
```

می بینیم که برنامه با افزایش تعداد بلوک ها تغییر نکرد. پس به نظر اجرای برنامه با دو بلوک ۱۰۲۴ تایی بهترین اجرا را دارد.

به دلیل اینکه اندازه ورودی زیاد است، برای تسریع دسترسی به حافظه `host`، استفاده از حافظه پین شده برای جلوگیری از paging توسط سیستم عامل به تسریع برنامه کمک می کند.

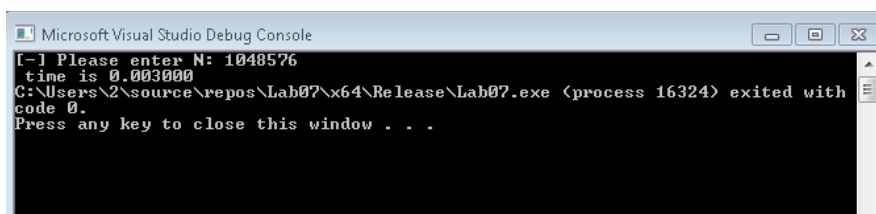
استفاده از `zero copy` این امکان را به نخبه های GPU میدهد که بتوانند به طور مستقیم به حافظه CPU دسترسی داشته باشند و دیگر نیازی به `copy` آن در داده های زیاد وجود نداشته باشد. به همین دلیل می تواند باعث تسریع در اجرای برنامه برنامه شود. اما استفاده از `zero copy` باعث به وجود آمدن محدودیت در پهنای باند حافظه می شود.

در مرحله بعدی، نوع داده را `float` در نظر می گیریم. ابتدا برای اطمینان از درستی برنامه را با ورودی ۳۲ امتحان می کنیم:



```
Microsoft Visual Studio Debug Console
[-] Please enter N: 32
time is 0.000000[-] array: 1.000000, 3.000000, 6.000000, 10.000000, 15.000000, 21.000000, 28.000000, 36.000000, 45.000000, 55.000000, 66.000000, 78.000000, 91.000000, 105.000000, 120.000000, 136.000000, 153.000000, 171.000000, 190.000000, 210.000000, 231.000000, 253.000000, 276.000000, 300.000000, 325.000000, 351.000000, 378.000000, 406.000000, 435.000000, 465.000000, 496.000000, 528.000000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 6236) exited with code 0.
Press any key to close this window . . .
```

اکنون با اندازه ورودی ۱۰۴۸۵۷۶ برنامه را بررسی می کنیم:

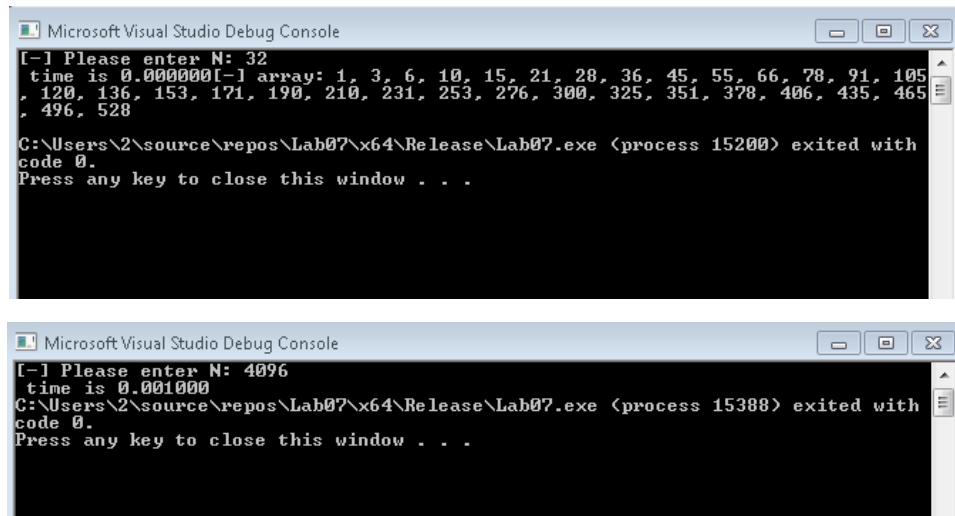


```
Microsoft Visual Studio Debug Console
[-] Please enter N: 1048576
time is 0.003000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 16324) exited with code 0.
Press any key to close this window . . .
```

می بینیم که زمان اجرای برنامه برابر با حالت `int` است.

در مرحله بعدی، یک کرنل دیگر برای اجرا با `shared memory` در نظر میگیریم که در کد پیوست به صورت کامنت شده قرار گرفته است. به جهت اینکه برای تخصیص حافظه مشترک نیاز به یک `const int` داریم، یک ثابت با نام `SIZE` ایجاد می

کنیم و تعداد ورودی را با آن مشخص می کنیم. به دلیل اینکه اندازه حافظه مشترک محدود است، نمی توان آن را ۱۰۴۸۵۷۶ اجرا کرد و به همین دلیل نمیتوان آن را با قسمت های قبلی نیز مقایسه کرد، زیرا افزایش بیش از حد حافظه باعث ایجاد ارور در برنامه می شود. به همین دلیل این قسمت را ابتدا با اندازه ۳۲ برای اطمینان از درستی برنامه و سپس با ۴۰۹۶ برای زمان، اجرا می کنیم:



```
Microsoft Visual Studio Debug Console
[+] Please enter N: 32
time is 0.000000[+] array: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351, 378, 406, 435, 465, 496, 528
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 15200) exited with code 0.
Press any key to close this window . . .

Microsoft Visual Studio Debug Console
[+] Please enter N: 4096
time is 0.001000
C:\Users\2\source\repos\Lab07\x64\Release\Lab07.exe (process 15388) exited with code 0.
Press any key to close this window . . .
```