Peak bandwidth)

نوع gpu با توجه به تصویر زیر Tesla T4 می باشد.

~~

```
There is 1 device supporting CUDA

Device 0: "Tesla T4"
  Major revision number:                         7
  Minor revision number:                         5
  Total amount of global memory:                 2958819328 bytes
  Number of multiprocessors:                     40
  Number of cores:                               320
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Clock rate:                                    1.59 GHz
  Concurrent copy and execution:                 Yes

TEST PASSED
```

در جدول مشخصات آن میبینیم که peak bandwidth برابر با 320GB/s می باشد.

Memory	
Memory Size:	16 GB
Memory Type:	GDDR6
Memory Bus:	256 bit
Bandwidth:	320.0 GB/s

توضیحات کرنل ها)

در تمامی اجرا ها اندازه ورودی برابر با $۲^{۲۴}$ یعنی ۱۶۷۷۷۲۱۶ می باشد. و تعداد نخ موجود در هر بلاک نیز در هر یک از سلسله مراتب محاسبات ۶۴ می باشد. در این روش سلسله مراتبی ابتدا اعداد ورودی را به $۲^{۱۸}$ بلاک تخصیص می دهیم. خروجی $۲^{۱۸}$ عدد جدید خواهد بود بار دیگر این اعداد را به $۲^{۱۲}$ بلاک می دهیم و خروجی به همین اندازه خواهیم داشت با تکرار بعدی اندازه خروجی برابر با $۲^{۶}$ یعنی همان ۶۴ می شود که در این قسمت صرفا از یک بلاک با ۶۴ نخ برای بدست آوردن جواب نهایی استفاده می کنیم. همچنین در این مراحل صرفا از دو آرایه A و B استفاده می کنیم که با توجه به این نکته که هر کرنل دقیقا از سایزی که مربوط به آن است استفاده می کند، مقادیر قبلی موجود در این آرایه ها مشکلی ایجاد نمی کند. پس در مرحله ۱ ورودی در A است و خروجی در B سپس در مرحله بعد ورودی در B است و خروجی در A. با توجه به همین نکته باید بتوانیم متوجه شویم که خروجی نهایی در کدام یک از آرایه ها قرار دارد تا آن را به هاست کپی کنیم.

به طور خلاصه تعداد بلوک ها و نخ ها در هر گام در جدول زیر قابل مشاهده است. (برای کرنل های ۱ و ۲ و ۳)

تعداد نخ در بلاک	تعداد بلاک	اندازه ورودی	شماره مرحله
۶۴	۲۶۲۱۴۴	۱۶۷۷۷۲۱۶	۱
۶۴	۴۰۹۶	۲۶۲۱۴۴	۲
۶۴	۶۴	۴۰۹۶	۳
۶۴	۱	۶۴	۴

به طور خلاصه تعداد بلوک ها و نخ ها در هر گام در جدول زیر قابل مشاهده است. (برای کرنل های ۴ و ۵)

تعداد نخ در بلاک	تعداد بلاک	اندازه ورودی	شماره مرحله
۶۴	۱۳۱۰۷۲	۱۶۷۷۷۲۱۶	۱
۶۴	۲۰۴۸	۱۳۱۰۷۲	۲
۶۴	۳۲	۲۰۴۸	۳
۳۲	۱	۳۲	۴

این کرنل روش کارش چند مرحله ای است و در هر مرحله با توجه به اندازه stride، فاصله بین دو خانه که باید با یکدیگر جمع شوند مشخص می شود. در این کد برای جمع یک حلقه for وجود دارد که داخل آن هر نخ با توجه به ایندکسش متوجه می شود که باید عمل جمع را انجام دهد یا نه. مشکل اصلی این روش واگرا بودن ایندکس نخ های مورد استفاده است. به عنوان مثال در دور اول نخ های ۰ و ۲ و ... دور بعدی ۰ و ۴ و ... و به همین ترتیب که باعث کاهش کارایی می شود.

خروجی در عکس زیر قابل مشاهده است.

```
[Reduction Using CUDA] - Starting...
GPU Device 0: "Tesla T4" with compute capability 7.5

[-] N = 16777216
Array size is 16777216
Computing result using CUDA Kernel...
Calculation elapsed time in msec = 1.861120
Total elapsed time in msec = 16.262239
```

۲)

تفاوت این کرنل با قبلی این است که در اینجا ایندکس ها را به طور مستقیم با استفاده از مقدار stride بدست می آوریم و در قسمت انجام کار آن هایی را که خارج از محدوده بلاک هستند بیکار می گذاریم. در نتیجه در این حالت در مرحله اول نخ های ۰ و ۱ و ... در مرحله دوم ۰ و ۱ و .. و به همین ترتیب ادامه می یابد. و دیگر مشکل واگرایی نخ ها را نخواهیم داشت.

خروجی در عکس زیر قابل مشاهده است.

```
[Reduction Using CUDA] - Starting...
GPU Device 0: "Tesla T4" with compute capability 7.5

[-] N = 16777216
Array size is 16777216
Computing result using CUDA Kernel...
Calculation elapsed time in msec = 1.818528
Total elapsed time in msec = 16.233408
```

۳)

تفاوت این کرنل با قبلی این است که در اینجا با خانه های پشت سر هم در حافظه کار می کنیم و دیگر مانند حالت قبل در هر مرحله اجرایی تمام حافظه را به دنبال تک خانه هایی که با آنها کار داریم نمی گردیم. یعنی در این حالت هر مرحله فقط با خانه های ۰ تا stride در حافظه کار داریم.(که با طرف مقابل جمع می شوند) تفاوت دیگر این روش این است که stride از زیاد به کم حرکت می کند و مقدار اولیه آن نصف اندازه بلاک است.

خروجی در عکس زیر قابل مشاهده است.

```
[Reduction Using CUDA] - Starting...
GPU Device 0: "Tesla T4" with compute capability 7.5

[-] N = 16777216
Array size is 16777216
Computing result using CUDA Kernel...
Calculation elapsed time in msec = 1.535136
Total elapsed time in msec = 15.836064
```

۴)

تفاوت این کرنل با قبلی این است که در اینجا پیش از وارد شدن به مراحل جمع اعداد، یکبار عمل جمع را انجام می دهیم. این کار باعث می شود که بتوانیم همان کار قبلی را با نصف کردن تعداد نخ ها (نصف شدن تعداد بلاک در نتیجه نصف شدن تعداد نخ ها) انجام دهیم که تسریع بسیار خوبی نیز داشته است. علت دیگر تسریع نیز دسترسی های کمتر به حافظه می باشد.

خروجی در عکس زیر قابل مشاهده است.

```
[Reduction Using CUDA] - Starting...
GPU Device 0: "Tesla T4" with compute capability 7.5

[-] N = 16777216
Array size is 16777216
Computing result using CUDA Kernel...
Calculation elapsed time in msec = 0.808928
Total elapsed time in msec = 15.390048
```

۵)

تفاوت این کرنل با قبلی این است که با توجه به این نکته که پردازنده های موجود در gpu داده های را با واحد warp برای اجرا استفاده می کنند و اندازه warp ۳۲ می باشد. در اینجا با توجه به اینکه زمان هایی که مقدار stride کوچک تر از ۳۲ است صرفا یک warp در حال اجرا داریم، برای آنها قسمت های اضافه کد را حذف کرده و مستقیما مقدارشان را دستی تعیین می کنیم. یعنی زمانی که مقدار stride در حلقه به ۳۲ برسد از آن خارج می شویم و ادامه کار را به صورت دستی انجام می دهیم.

خروجی در عکس زیر قابل مشاهده است.

```
[Reduction Using CUDA] - Starting...
GPU Device 0: "Tesla T4" with compute capability 7.5

[-] N = 16777216
Array size is 16777216
Computing result using CUDA Kernel...
Calculation elapsed time in msec = 0.417920
Total elapsed time in msec = 14.729344
```

جدول نتایج)

جدول نتایج شامل مجموعه اطلاعات جدول صفحه ۲۳ به همراه کل زمان اجرا به صورت زیر می باشد.

	Calculation Time	Bandwidth	Step Speedup	Cumulative Speedup	Total Time
Kernel 1	1.861 ms	36.060 GB/s			16.262
Kernel 2	1.818 ms	36.913 GB/s	1.023x	1.023x	16.233
Kernel 3	1.535 ms	43.719 GB/s	1.184x	1.212x	15.836
Kernel 4	0.809 ms	82.952 GB/s	1.897x	2.300x	15.390
Kernel 5	0.417 ms	160.932 GB/s	1.940x	4.462x	14.729

مقایسه اجرا روی cpu و gpu)

نتیجه اجرا کد سریال روی cpu برای همان ورودی 2^{24} به صورت زیر می باشد.

```
[Reduction Using CUDA] - Starting...
GPU Device 0: "Tesla T4" with compute capability 7.5

[-] N = 16777216
Array size is (16777216, 1)
Computing result using CUDA Kernel...
Elapsed time in msec = 43.871647
```

ورودی مان کاملا بزرگ است و برای مقایسه مناسب است. همان طور که مشخص است زمان اجرای سریال روی cpu در حدود ۴۴ ms می باشد. در حالی که بهترین زمان اجرا روی gpu حدود ۰.۵ ms می باشد که اختلاف بسیار زیادیست. اگر زمان انتقال داده ها را هم در نظر بگیریم زمان کاری gpu برابر با حدود ۱۵ ثانیه خواهد بود. که در حد 3 / 1 زمان اجرا کد سریال می باشد که باز هم خیلی نتیجه بهتری است. حالت منصفانه این است که واقعا زمان انتقال را نیز با زمان اجرا در gpu جمع کنیم. چون تا زمانی که ورودی را به gpu ندهیم توان شروع اجرا را ندارد. همچنین اگر نتیجه را نیز بدست آورد و آن را برنگردانیم باز هم بی فایده خواهد بود. در نتیجه به صورت کلی می توان گفت gpu حدودا ۳ برابر بهتر از cpu عمل می کند. مشخصا اگر اندازه ورودی کم باشد سربار انتقال داده به gpu باعث می شود تا نتایج cpu از لحاظ زمانی بهتر باشند.