دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

برنامه‌نویسی چندهسته‌ای

تمرین ششم: برنامه‌نویسی CUDA

رادین شایانفر

شماره دانشجویی: ۹۷۳۱۰۳۲

بهار ۱۴۰۰

حداکثر پهنای باند حافظه: در این تمرین، از کارت گرافیک با Compute Capability برابر ۶.۱ استفاده شده است. در شکل زیر قسمتی از خروجی کد deviceQuery را روی آن می‌بینیم.

```
Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce GTX 1060 6GB"
  CUDA Driver Version / Runtime Version          11.3 / 11.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 6144 MBytes (64
  (10) Multiprocessors, (128) CUDA Cores/MP:     1280 CUDA Cores
  GPU Max Clock rate:                            1785 MHz (1.78
  Memory Clock rate:                             4004 Mhz
  Memory Bus Width:                              192-bit
  L2 Cache Size:                                 1572864 bytes
```

از آنجا که حافظه Double Data Rate (DDR) است، پهنای باند حافظه برابر است با:

$$\frac{192}{8} \times \ 4004\text{MHz} \times 2 \ = 192.192 \ {}^{\text{GB}}/_{\text{S}}$$

برنامه سریال: برنامه سریال در فایل serial.c نوشته شده است. زمان اجرای برنامه را به ازای ورودی‌های مختلف اندازه‌گیری کرده و در جدول زیر وارد می‌کنیم.

اندازه ورودی	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)
2^{22}	0.92	16.98
2^{26}	14.23	17.56
2^{30}	200.55	19.94

کد چندهسته‌ای روی GPU داخل فایل kernel.cu قرار دارد. در این فایل در ابتدای برنامه ماکرو KERNEL_NUM مشخص کننده کرنل اجرا شونده است که مقادیر ۱ تا ۵ را می‌گیرد. در جداول زیر اندازه بلوک در تمامی قسمت‌ها برابر ۲۵۶ است.

برای کرنل‌های مختلف تعداد kernel launchها و پارامترهای آن‌ها متفاوت است. برای هر یک از اندازه‌های ورودی در **۳ کرنل اول** پارامترهای آن مقادیر زیر هستند.

```
n: 2^22
blocks: (256, 1, 1), grid(16384, 1, 1)
blocks: (256, 1, 1), grid(64, 1, 1)
blocks: (64, 1, 1), grid(1, 1, 1)
```

```
n: 2^26
blocks: (256, 1, 1), grid(262144, 1, 1)
blocks: (256, 1, 1), grid(1024, 1, 1)
blocks: (256, 1, 1), grid(4, 1, 1)
blocks: (4, 1, 1), grid(1, 1, 1)
```

```
n: 2^30
blocks: (256, 1, 1), grid(4194304, 1, 1)
blocks: (256, 1, 1), grid(16384, 1, 1)
blocks: (256, 1, 1), grid(64, 1, 1)
blocks: (64, 1, 1), grid(1, 1, 1)
```

برای **دو کرنل آخر** نیز پارامترها به شکل زیر هستند.

```
n: 2^22
blocks: (256, 1, 1), grid(8192, 1, 1)
blocks: (256, 1, 1), grid(16, 1, 1)
blocks: (8, 1, 1), grid(1, 1, 1)
```

```
n: 2^26
blocks: (256, 1, 1), grid(131072, 1, 1)
blocks: (256, 1, 1), grid(256, 1, 1)
blocks: (128, 1, 1), grid(1, 1, 1)
```

```
n: 2^30
blocks: (256, 1, 1), grid(2097152, 1, 1)
blocks: (256, 1, 1), grid(4096, 1, 1)
blocks: (256, 1, 1), grid(8, 1, 1)
blocks: (4, 1, 1), grid(1, 1, 1)
```

در ادامه توضیح هر یک از کرنل‌ها به همراه نتایج آن‌ها آمده است.

۱. در این کرنل به تعداد المان‌های آرایه، نخ اختصاص می‌دهیم. سپس در ابتدا هر نخ المان متناظر خود را در حافظه اشتراکی لود می‌کند. پس از آن در هر مرحله نخ‌های با اندیس به شکل 2^k (k برابر مرحله اجرای حلقه) یک عمل جمع انجام می‌دهند و دیگر نخ‌ها عملی انجام نمی‌دهند. در انتها نخ شماره صفر مقدار reduce شده آن بلوک را در آرایه خروجی می‌نویسد.

اندازه ورودی	زمان اجرای کل (میلی ثانیه)	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)	تسریع کل
2^{22}	3.77	1.69	9.23	0.24
2^{26}	38.41	15.47	16.15	0.37
2^{30}	570.87	226.65	17.64	0.35

۲. در این کرنل به جای اینکه نخ‌ها را از بین نخ‌های فعال حذف کنیم، نخ‌ها را از انتها در هر مرحله نصف می‌کنیم. این کار باعث می‌شود واگرایی کمتر شود و نخ‌های داخل یک warp بیشتر کار مشابه انجام دهند.

اندازه ورودی	زمان اجرای کل (میلی ثانیه)	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)	تسریع محاسبات گام	تسریع محاسبات تجمعی	تسریع کل
2^{22}	3.51	1.37	11.35	1.23	1.23	0.26
2^{26}	33.32	10.82	23.10	1.42	1.42	0.42
2^{30}	493.78	150.06	26.65	1.51	1.51	0.40

۳. با معکوس کردن stride در این کرنل (یعنی از مقدار بزرگ‌تر به سمت یک رفتن) دسترسی به حافظه نخ‌ها پشت سر هم می‌شود و باعث بهبود عملکرد می‌شود.

اندازه ورودی	زمان اجرای کل (میلی ثانیه)	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)	تسریع محاسبات گام	تسریع محاسبات تجمعی	تسریع کل
2^{22}	3.14	1.09	14.30	1.25	1.55	0.29
2^{26}	31.49	9.03	27.67	1.19	1.71	0.45
2^{30}	472.13	127.86	31.28	1.17	1.77	0.42

۴. در کرنل چهارم پیش از ریختن المان‌ها در حافظه مشترک، یک جمع انجام می‌دهیم و سپس این عمل جمع را داخل حافظه مشترک می‌نویسیم. این کار باعث می‌شود تعداد نخ‌ها در هر مرحله با افزودن تنها یک دستورالعمل load و add نصف شود و در نتیجه زمان اجرا بهبود پیدا می‌کند.

اندازه ورودی	زمان اجرای کل (میلی ثانیه)	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)	تسریع محاسبات گام	تسریع محاسبات تجمعی	تسریع کل
2^{22}	3.12	1.06	14.63	1.02	1.59	0.29
2^{26}	28.15	5.66	44.13	1.59	2.73	0.50
2^{30}	425.67	83.19	48.08	1.53	2.72	0.47

۵. در این کرنل گفته شده از آنجا که ۶ اجرای آخر حلقه همگی در یک warp انجام می‌دهند، دیگر نیازی به بقیه warpها و همچنین `__syncthreads()` نیست. با این حال طبق مطلب سایت nvidia[1] از نسخه ۹ کودا دیگر فرض همگام بودن نخ‌های یک warp امن نیست. در نتیجه **این کرنل به درستی کار نمی‌کند** (به هر روی کد این کرنل با افزودن `__syncthreads()` در تابع `wrongKernel5` نوشته شده است و نتایج آن در جدول زیر آمده است).

اندازه ورودی	زمان اجرای کل (میلی ثانیه)	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)	تسریع محاسبات گام	تسریع محاسبات تجمعی	تسریع کل
2^{22}	3.06	0.89	17.39	1.19	1.89	1.03
2^{26}	26.79	4.22	59.15	1.34	3.66	0.53
2^{30}	413.10	67.13	59.58	1.23	3.37	0.48

[1] https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/

در اینجا می‌توان کاری مشابه کار گفته شده را انجام داد. از آنجا که اگر شرط داخل حلقه برای یک نخ false شود تا آخر اجرای آن نخ false می‌ماند، می‌توان در قسمت else شرط عبارت ;return گذاشت تا اجرای این نخ‌ها در همانجا پایان یابد. این کار کمک می‌کند تا warpهایی که کارشان تمام شده زودتر منابع را آزاد کنند و زمان اجرا کاهش می‌یابد.

اندازه ورودی	زمان اجرای کل (میلی ثانیه)	زمان محاسبات (میلی ثانیه)	پهنای باند (GB/s)	تسریع محاسبات گام	تسریع محاسبات تجمعی	تسریع کل
2^{22}	3.05	0.86	18.05	1.23	1.96	1.06
2^{26}	26.27	4.12	60.60	1.37	3.75	0.54
2^{30}	418.76	74.97	53.35	1.1	3.02	0.47

اگرچه می‌بینیم که زمان انجام محاسبات بر روی GPU بسیار نسبت به حالت سریال روی CPU سریع‌تر است، اما زمان اجرای کل روی GPU (شامل کپی کردن داده‌ها) باز هم نسبت به CPU کندتر است. از آنجا که مقایسه منصفانه، مقایسه‌ای است که **کل زمان اجرا** را مقایسه کند، در نتیجه به نظر می‌رسد با این اندازه مسئله‌های بررسی شده سربار بردن محاسبات روی GPU به صرفه نیست.