تمرین درس پردازش موازی سری چهارم

نام: سینا قادرمرزی شماره‌ی دانشجویی: ۹۳۱۳۱۰۳۸

در این تمرین، ضرب دو ماتریس با استفاده از CUDA انجام گرفته است.

روش استفاده شده

در این تمرین همه‌ی پردازش بر روی یک thread block انجام گرفته و بنابراین محدود به استفاده از 1024 عدد CUDA thread هستیم. پیاده سازی در دو مرحله (با ورودی محدود و با ورودی نامحدود) انجام گرفته است و در هر مرحله شامل دو حالت thread یک بعدی و دو بعدی است. در هر حالت برنامه شامل ضرب یک ماتریس N×N (A) در یک ماتریس N×N (B) می‌باشد که در نتیجه ماتریس N×N (C) به دست می‌آید. عدد N به همراه تعداد thread ها به عنوان Commandline Argument به برنامه داده می‌شود.

مرحله‌ی اول)

شرایط ورودی:

- حداکثر مقدار N برابر ۳۲ است
- تعداد thread ها برابر N^2 می‌باشد.

در مرحله‌ی اول هریک از عناصر ماتریس حاصلضرب (C) توسط یک CUDA thread محاسبه می‌شود. بنابر این حداکثر تعداد عناصر ماتریس خروجی (عدد N^2) برابر ۱۰۲۴ می‌باشد. اگر حالت دو بعدی را در نظر بگیریم هر CUDA thread دارای یک بعد x و یک بعد y می‌باشد. و thread(x,y) عنصر سطر x و ستون y از ماتریس C را محاسبه می‌نماید.

```
__global__ void mutrixMulKernel(const int *m_d,const int *n_d, int *p_d, int N)
{
	int pvalue = 0;
	int y = threadIdx.y;
	int x = threadIdx.x;
	for (int k = 0; k < N; k++)
	{
		int melement = m_d[y * N + k];
		int nelement = n_d[k * N + x];
		pvalue += melement * nelement;
	}
	p_d[y * width + x] = pvalue;
}
```

برای تبدیل این حالت به یک‌بعدی کافی است که مختصات thread (مختصات عنصری از C که قرار است توسط thread محاسبه شود) را از روی اندیس یک‌بعدی آن تولید کنیم.

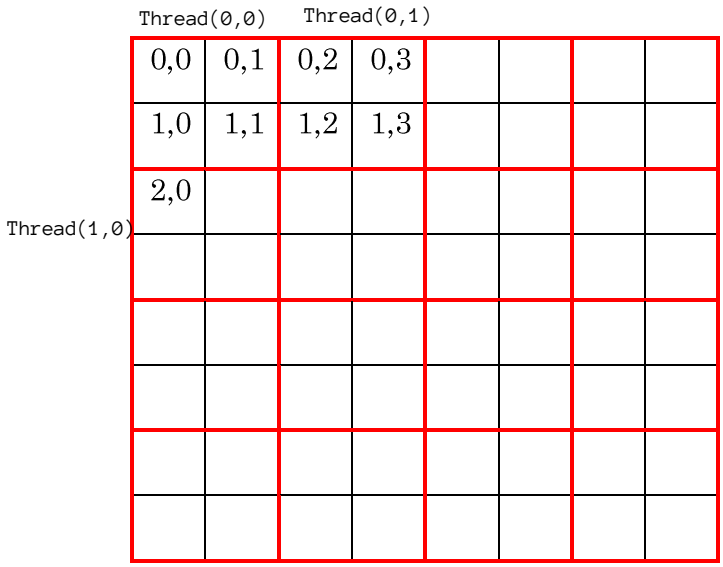
```
__global__ void mutrixMulKernel(const int *m_d,const int *n_d, int *p_d, int N)
{
	int pvalue = 0;
	int y = threadIdx.x / N;
	int x = threadIdx.x % N;
	for (int k = 0; k < N; k++)
	{
		int melement = m_d[y * N + k];
		int nelement = n_d[k * N + x];
		pvalue += melement * nelement;
	}
	p_d[y * width + x] = pvalue;
}
```

حالت دوم)

شرایط ورودی:

- تعداد thread های وارد شده باید مکعب کامل باشد.
- N باید کوچکتر از ۱۰۲۴ و بر جذر تعداد thread ها بخش‌پذیر باشد.

در این حالت با توجه به این که فرضی در مورد ابعاد ماتریس نداریم هر یک از threadها باید بیش از یک عنصر از ماتریس C را محاسبه نماید. به این منظور ماتریس C را به بلوک‌هایی تقسیم می‌کنیم که هریک از بلوک‌ها توسط یک thread محاسبه می‌گردند.



نتایج و تحلیل

این برنامه بر روی یک سیستم با مشخصات زیر اجرا گردیده است.

CPU : Intel Core i7 4702MQ

GPU: Nvidia GeForce GT 740M

مقادیر speedup نسبت به الگوریتم serial به شرح زیر است. لازم به ذکر است برای داده‌های با N>512 برنامه crashکرده و درایور GPU ریست می‌شود. که ظاهراً به دلیل محدودیتی است که برروی زمان اجرای کرنل‌ها اعمال شده است. بنابراین تسریع‌ها برای داده‌های کوچکتر از این مقدار گزارش شده است.

threads	N= 32	N= 64	N= 128	N= 256	N= 512	N= 1024
512	0	0	1	7	-	?
1024	0	0	1	2.33	5.5	?

با توجه به کوچک بودن اندازه‌ی داده‌ها (و عدام امکان بررسی برای داده‌های بزرگتر) به نظر می‌رسد که نتایج به دست آمده به خوبی نمایانگر تسریعی می‌توان به آن دست یافت نمی‌باشد.

اقداماتی که جهت بهبود این برنامه می‌توان انجام داد.

۱- استفاده از Thread Block های متعدد جهت استفاده از همه‌ی هسته‌های پردازشی GPU

۲- استفاده از Shared Memory می‌تواند برای Thread های داخل یک Block سرعت را بسیار افزایش دهد.