۱ گام اول

در این گام، کد ضرب ماتریس‌هایی با حداکثر اندازه ۳۲ در ۳۲ را می‌نویسیم.

در صورتی که در ابتدای برنامه خط

```
#define DEBUG
```

قرار گیرد، نتایج محاسبات پس از اجرای کرنل و پیش از آزادسازی حافظه‌ها بر روی میزبان (CPU) به شکل تک‌نخی بررسی می‌شود. اگر ضرب ماتریس‌ها نادرست انجام شده باشد، در این قسمت پیغام wrong answer چاپ می‌شود.

۲ گام دوم

در این گام سه راه‌حل مختلف برای ضرب ماتریس‌ها پیاده‌سازی شده است. در روش اول تنها ۱۰۲۴ نخ در یک بلوک تمام محاسبات را انجام می‌دهند. این روش در کرنل matMulA1Kernel پیاده‌سازی شده است. روش دوم که در کرنل matMulA2Kernel پیاده شده است، به تعداد درایه‌های ماتریس‌ها نخ در بلوک‌های مختلف اختصاص می‌دهد. در روش سوم و آخر از تکنیک tiling با اندازه بلوک و کاشی‌های ۳۲ در ۳۲ استفاده شده است. روش سوم نیز در کرنل matMulA3Kernel آمده است.

در ابتدای برنامه با تغییر خط

```
#define APPROACH x
```

می‌توان برنامه را با راه‌حل شماره x اجرا کرد.

حال برنامه را با سه روش گفته شده (به همراه کد سریال روی CPU) می‌سنجیم. نتایج در جدول ۱ آمده است. زمان‌های اجرا میانگین چند بار اجرا و تسریع، میانگین تسریع دو ستون آخر است.

همانطور که می‌بینیم روش اول تسریع کمی دارد زیرا به خوبی از تمام توان پردازشی GPU استفاده نمی‌کند. در روش دوم زمان اجرا بهبود بسیاری دارد. روش سوم نیز به علت استفاده بهتر از حافظه کش، زمان آن اندکی بهبود پیدا کرده است.

جدول ۱: زمان‌های اجرا (میلی ثانیه) در سه روش اول

موازی‌سازی	اندازه ورودی			تسریع
	2^{10}	2^{11}	2^{12}	
سریال	۱۷۵۳/۲	۵۶۲۸۰/۰	۵۳۸۶۵۱/۵	–
راه‌حل اول	۶۸۷/۸	۶۰۱۱/۷	۵۳۴۵۵/۷	۹/۷۱
راه‌حل دوم	۸/۵	۶۸/۷	۶۲۲/۰	۸۴۲/۶۰
راه‌حل سوم (کاشی‌کاری)	۸/۸	۶۹/۹	۵۹۲/۸	۸۵۶/۹۰

۳ گام سوم

در این گام، راه‌حل چهارم (کاشی‌کاری با حافظه مشترک) را در کرنل matMulA4Kernel پیاده‌سازی می‌کنیم. نتیجه اجرای آن در جدول ۲ آمده است.

جدول ۲: زمان‌های اجرا (میلی ثانیه) در روش چهارم (کاشی‌کاری با حافظه مشترک)

موازی‌سازی	اندازه ورودی			تسریع
	2^{10}	2^{11}	2^{12}	
کاشی‌کاری با حافظه مشترک	۸/۲	۶۱/۵	۵۱۵/۳	۹۸۰/۲۱

مطابق انتظار بهبودهای بیشتری در سرعت اجرا نسبت به راه‌حل سوم (استفاده نکردن از حافظه مشترک و استفاده از مکانیزم‌های سخت‌افزاری کش) در این روش داریم.