این الگوریتم برای ضرب دو ماتریس با استفاده از روش‌های CPU و GPU پیاده‌سازی شده است. الگوریتم به شرح زیر عمل می‌کند:

1. در ابتدا، اندازه ماتریس (matrix_size) را تعیین می‌کنیم و دو ماتریس مربوطه (matrix_a و matrix_b) را به اندازه matrix_size × matrix_size با استفاده از تابع np.random.rand ایجاد می‌کنیم. این تابع مقادیر تصادفی بین 0 و 1 را در ماتریس تولید می‌کند.
2. تابع cpu_matrix_multiplication به عنوان الگوریتم ضرب ماتریس با استفاده از CPU پیاده‌سازی شده است. در این تابع، زمان شروع اجرای الگوریتم (start_time) را ثبت می‌کنیم.

- ماتریس‌های ورودی (matrix_a و matrix_b) را با استفاده از تابع np.dot ضرب می‌کنیم و نتیجه را در متغیر cpu_result ذخیره می‌کنیم.
- زمان پایان اجرای الگوریتم (end_time) را ثبت می‌کنیم و زمان اجرا را با اختلاف end_timeو start_time محاسبه می‌کنیم.
- نتیجه ضرب ماتریس (cpu_result) و زمان اجرا (execution_time) را برمی‌گردانیم.

3. تابع gpu_matrix_multiplication نیز همان عملکرد ضرب ماتریس را با استفاده از GPU پیاده‌سازی می‌کند. در این تابع:

- ماتریس‌های ورودی (matrix_a و matrix_b) را به فرمت قابل قبول برای GPU تبدیل می‌کنیم و در متغیرهای gpu_matrix_a و gpu_matrix_b ذخیره می‌کنیم.
- با استفاده از تابع cp.dot ، ضرب ماتریس را با استفاده از GPU انجام می‌دهیم و نتیجه را در متغیر gpu_result ذخیره می‌کنیم.
- سپس نتیجه را به فرمت قابل قبول برای CPU تبدیل کرده و در متغیر cpu_result ذخیره می‌کنیم.
- زمان شروع و پایان اجرای الگوریتم را ثبت کرده و زمان اجرا را محاسبه می‌کنیم.
- نتیجه ضرب ماتریس (cpu_result) و زمان اجرا (execution_time) را برمی‌گردانیم.

4. در حلقه for با تعداد تکرار 5 بار، مراحل زیر تکرار می‌شوند:

- در هر مرحله، شماره مرحله (i+1) را نمایش می‌دهیم.

- با استفاده از تابع psutil.cpu_percent ، درصد استفاده کلی از CPU را در متغیر cpu_percentذخیره می‌کنیم.
- با استفاده از تابع psutil.cpu_percent با پارامتر percpu=True ، درصد استفاده از هر هسته از CPU را در لیست cpu_percent_per_core ذخیره می‌کنیم.
- مشخصات مربوط به استفاده از پردازنده گرافیکی را با استفاده از تابع gpustat.new_query نمایش می‌دهیم.
- ابتدا الگوریتم ضرب ماتریس با استفاده از روش CPU اجرا می‌شود و نتیجه ضرب و زمان اجرا را در متغیرهای cpu_result و cpu_execution_time ذخیره می‌کنیم.
- سپس الگوریتم ضرب ماتریس با استفاده از روش GPU اجرا می‌شود و نتیجه ضرب و زمان اجرا را در متغیرهای gpu_result و gpu_execution_time ذخیره می‌کنیم.
- در صورت نیاز، نتایج حاصل از ضرب ماتریس با روش‌های CPU و GPU را نمایش می‌دهیم.
- در نهایت، زمان اجرا برای هر دو روش را نمایش می‌دهیم.

این کد برای مقایسه زمان اجرا و استفاده از منابع سخت‌افزاری بین روش‌های CPU و GPU در عملیات ضرب ماتریس مورد استفاده قرار می‌گیرد.

با توجه به ، می‌توانیم تحلیلی بر روی آن‌ها انجام دهیم:

1. مشخصات استفاده از پردازنده مرکزی (CPU) و پردازشگر گرافیکی (GPU) در هر مرحله نمایش داده می‌شود. می‌توان توجه کرد که مقدار استفاده از پردازنده مرکزی در هر مرحله متفاوت است و ممکن است به دلایل مختلفی مانند بارکاری سیستم تغییر کند.
2. نتیجه ضرب ماتریس برای روش‌های CPU و GPU به صورت یکسان در همه مراحل نمایش داده می‌شود. این نتایج باید یکسان باشند زیرا عمل ضرب ماتریس با استفاده از هر دو روش صورت می‌گیرد و باید به نتایج یکسانی منجر شود.
3. زمان اجرای الگوریتم برای هر دو روش (CPU و GPU) نمایش داده می‌شود. می‌توان توجه کرد که در هر مرحله، زمان اجرای الگوریتم برای روش GPU کمتر از زمان اجرای الگوریتم برای روش CPU است. این نشان می‌دهد که استفاده از پردازشگر گرافیکی (GPU) بهبود قابل توجهی در سرعت اجرای عملیات ضرب ماتریس دارد.

4. استفاده از پردازنده مرکزی (CPU) در همه مراحل بالا استفاده بیشتری نسبت به پردازشگر گرافیکی (GPU)دارد. این ممکن است به دلیل استفاده‌های دیگر از پردازنده مرکزی در سیستم باشد، اما ممکن است نیاز به بهینه‌سازی بیشتر در استفاده از GPU و استفاده بهینه‌تر از قابلیت‌های آن باشد.

5. مشاهده می‌شود که در مرحله 3، مقدار استفاده از پردازنده مرکزی (CPU) نسبت به مراحل قبلی بیشتر است و همچنین زمان اجرای الگوریتم برای هر دو روش (CPU و GPU) نسبت به مراحل قبلی افزایش یافته است. این می‌تواند به دلیل بارکاری سیستم در آن مرحله باشد و نشان می‌دهد که می‌تواند باعث افزایش زمان اجرای الگوریتم باشد.

با توجه به تحلیل فوق، می‌توان نتیجه گرفت که استفاده از پردازشگر گرافیکی (GPU) در عملیات ضرب ماتریس می‌تواند زمان اجرا را بهبود بخشیده و سرعت اجرای الگوریتم را افزایش دهد. همچنین، بارکاری سیستم و استفاده از منابع سخت‌افزاری (مانند پردازنده مرکزی) می‌تواند تأثیر قابل توجهی بر زمان اجرا داشته باشد و باید در بررسی‌های بعدی در نظر گرفته شود.

```
matrix_size = 100:
Overall CPU usage: 31.4%
Per-core CPU usage: [29.7, 33.2]%
GPUStatCollection(host=d6537d5522b1, [
  [0] Tesla T4         | 74°C,   0 % |   653 / 15360 MB |
])
CPU Execution Time: 0.004936218261718750 seconds
GPU Execution Time: 0.0008718967437744141 seconds
--------------------------------------------------
matrix_size = 500:
Overall CPU usage: 100.0%
Per-core CPU usage: [100.0, 100.0]%
GPUStatCollection(host=d6537d5522b1, [
  [0] Tesla T4         | 74°C,   0 % |   653 / 15360 MB |
])
CPU Execution Time: 0.008203983306884766 seconds
GPU Execution Time: 0.006677150726318359 seconds
--------------------------------------------------
matrix_size = 1000:
Overall CPU usage: 72.4%
Per-core CPU usage: [71.4, 78.6]%
GPUStatCollection(host=d6537d5522b1, [
  [0] Tesla T4         | 74°C,   0 % |   653 / 15360 MB |
])
CPU Execution Time: 0.055381536483764650 seconds
GPU Execution Time: 0.036133527755737305 seconds
--------------------------------------------------
matrix_size = 2000:
Overall CPU usage: 75.6%
Per-core CPU usage: [78.3, 70.8]%
GPUStatCollection(host=d6537d5522b1, [
  [0] Tesla T4         | 74°C,   2 % |   653 / 15360 MB |
])
CPU Execution Time: 0.43175935745239260 seconds
GPU Execution Time: 0.21144556999206543 seconds
--------------------------------------------------
matrix_size = 3000:
Overall CPU usage: 81.4%
Per-core CPU usage: [86.7, 75.9]%
GPUStatCollection(host=d6537d5522b1, [
  [0] Tesla T4         | 74°C,  36 % |   653 / 15360 MB |
])
CPU Execution Time: 1.3821666240692139 seconds
GPU Execution Time: 0.37188124656677246 seconds
--------------------------------------------------
```

```
CPU Result:
[[242.02755221 234.65317101 243.81338223 ... 253.03609432 253.39417444
  239.14111004]
 [235.61422378 235.34902061 241.49263317 ... 244.00526631 244.24091553
  234.91872903]
 [247.82741382 245.46272466 256.15395454 ... 262.80338552 257.62121992
  239.1000921 ]
 ...
 [236.80267019 236.92460775 243.41677306 ... 251.83664832 250.84095005
  241.59202376]
 [243.98133577 237.1846115  243.18618403 ... 250.32889244 245.93479783
  238.33515505]
 [239.04345252 240.25968763 246.89495836 ... 252.45104164 253.47848923
  234.53080361]]
GPU Result:
[[242.02755221 234.65317101 243.81338223 ... 253.03609432 253.39417444
  239.14111004]
 [235.61422378 235.34902061 241.49263317 ... 244.00526631 244.24091553
  234.91872903]
 [247.82741382 245.46272466 256.15395454 ... 262.80338552 257.62121992
  239.1000921 ]
 ...
 [236.80267019 236.92460775 243.41677306 ... 251.83664832 250.84095005
  241.59202376]
 [243.98133577 237.1846115  243.18618403 ... 250.32889244 245.93479783
  238.33515505]
 [239.04345252 240.25968763 246.89495836 ... 252.45104164 253.47848923
  234.53080361]]
```