

## 令和6年度 深層学習

# 二次元行列同士の積の計算と二次元行列の畳み込みにおける 行列サイズと実行時間の関係と、 python, numpy, pyTorch の比較

令和7年4月24日

大阪国際工科専門職大学

工科学部

情報工学科

AI 開発コース

OK240100 藤村勇仁

## 目次

1	概要	1
2	行列の積	1
2.1	プログラム . . . . .	1
2.2	実行結果 . . . . .	1
2.3	考察 . . . . .	2
3	行列の畳み込み	3
3.1	プログラム . . . . .	3
3.2	実行結果 . . . . .	3
3.3	考察 . . . . .	4
付録 A	2次元行列同士の積を求めるプログラム	5
付録 B	2次元行列の畳み込みを求めるプログラム	9

## 1 概要

本課題では、2つの正方行列の積と正方行列と $3 \times 3$ 行列の畳み込みそれぞれを実行するpythonプログラムを作成して、それぞれの行列サイズと実行時間の関係を調べる。また、python, numpy, pytorchの3つのライブラリを用いて、行列の積と畳み込みを行い、実行時間を比較する。

## 2 行列の積

2つの正方行列の積の計算のサイズと実行時間の関係を調べ、python, numpy, pyTorchの比較を行う。行列のサイズは200から200ずつ増加させ、pythonだけでの計算時間が60秒を超えたら1000ずつ増加させる。行列のサイズが大きくなるにつれて、実行時間がどのように変化するかを調べる。

### 2.1 プログラム

コード付録Aに、行列の積を計算するpythonプログラムを示す。

### 2.2 実行結果

コード1と、図1に、行列の積の計算結果を示す。

コード1 付録付録Aの実行結果

---

1	Size: 200,	Python: 0.6059s,	NumPy: 0.0020s,	Cupy: 0.0260s,	PyTorch: 0.0009s
2	Size: 400,	Python: 5.2042s,	NumPy: 0.0025s,	Cupy: 0.0011s,	PyTorch: 0.0000s
3	Size: 600,	Python: 20.5605s,	NumPy: 0.0039s,	Cupy: 0.0018s,	PyTorch: 0.0010s
4	Size: 800,	Python: 53.1748s,	NumPy: 0.0060s,	Cupy: 0.0030s,	PyTorch: 0.0020s
5	Size: 1800,	Python: -----,	NumPy: 0.0476s,	Cupy: 0.0179s,	PyTorch: 0.0180s
6	Size: 2800,	Python: -----,	NumPy: 0.1761s,	Cupy: 0.0585s,	PyTorch: 0.0585s
7	Size: 3800,	Python: -----,	NumPy: 0.3741s,	Cupy: 0.1394s,	PyTorch: 0.1396s
8	Size: 4800,	Python: -----,	NumPy: 0.6643s,	Cupy: 0.2409s,	PyTorch: 0.2172s
9	Size: 5800,	Python: -----,	NumPy: 1.2572s,	Cupy: 0.2860s,	PyTorch: 0.2317s
10	Size: 6800,	Python: -----,	NumPy: 1.8989s,	Cupy: 0.3463s,	PyTorch: 0.3224s
11	Size: 7800,	Python: -----,	NumPy: 3.0902s,	Cupy: 0.1120s,	PyTorch: 0.3983s
12	Size: 8800,	Python: -----,	NumPy: 3.8734s,	Cupy: 0.1656s,	PyTorch: 0.1691s
13	Size: 9800,	Python: -----,	NumPy: 5.4696s,	Cupy: 0.2353s,	PyTorch: 0.3565s
14	Size: 10800,	Python: -----,	NumPy: 12.1399s,	Cupy: 0.3184s,	PyTorch: 0.5472s
15	Size: 11800,	Python: -----,	NumPy: 9.9611s,	Cupy: 0.3828s,	PyTorch: OOM

---

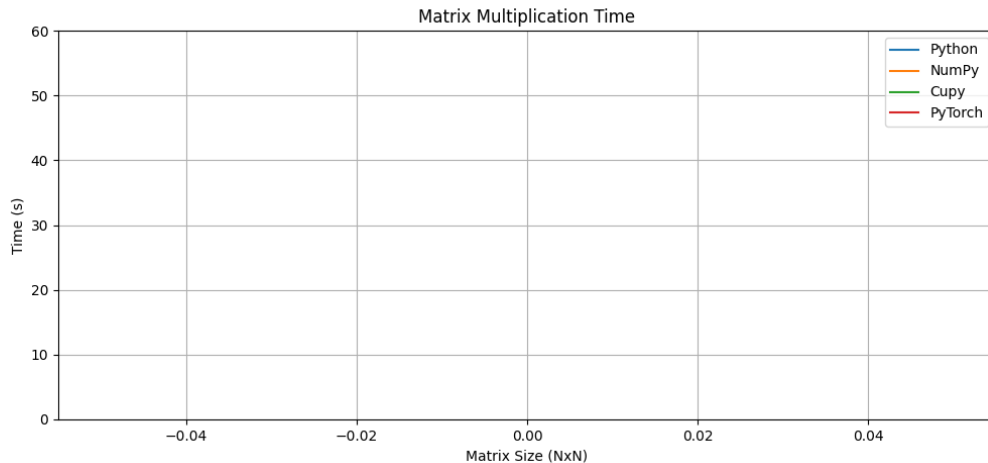


図1 付録付録 A の実行結果

## 2.3 考察

実行結果より、Python、NumPy、CuPy、PyTorch のいずれの方法も、理論上は  $O(n^3)$  に近い計算量であることがわかる。しかし、実際の計算時間としては  $\text{Python} > \text{NumPy} > \text{CuPy} \approx \text{PyTorch}$  となっており、これは NumPy や CuPy、PyTorch が内部的に BLAS (Basic Linear Algebra Subprograms) ライブラリを使用しており、高速化されているためである。

BLAS とは、線形代数の基本的な演算（行列積、ベクトル演算など）を高速に実行するための標準ライブラリで、CPU のキャッシュやパイプラインの効率的な活用、マルチスレッドによる並列化といった最適化が行われている。

実際に、NumPy で使用されている BLAS 実装を確認するために `numpy.__config__.show()` を実行すると、以下のような出力が得られる：

---

```

1 "blas": {
2   "name": "scipy-openblas",
3   "found": true,
4   "version": "0.3.27",
5   "detection method": "pkgconfig",
6   "include directory": "C:/Users/runneradmin/AppData/Local/Temp/cibw-run-4yyn90e0/cp312-
   win_amd64/build/venv/Lib/site-packages/scipy_openblas64/include",
7   "lib directory": "C:/Users/runneradmin/AppData/Local/Temp/cibw-run-4yyn90e0/cp312-
   win_amd64/build/venv/Lib/site-packages/scipy_openblas64/lib",
8   "openblas configuration": "OpenBLAS 0.3.27 USE64BITINT DYNAMIC_ARCH NO_AFFINITY Haswell
   MAX_THREADS=24",
9   "pc file directory": "D:/a/numpy/numpy/.openblas"
10 },

```

---

このように、使用されている BLAS ライブラリが OpenBLAS であることが確認できる。

また、CuPy や PyTorch は、明示的に GPU を使用する設定をすることで、行列積などの処理を GPU 上で実行することができる。GPU 内部ではメモリ帯域幅が広く、大量の演算ユニットを活用した並列計算が可能であるため、CPU と比較して高速な処理が実現される。

その結果、同じアルゴリズム的には  $O(n^3)$  の処理でも、ハードウェアとライブラリの違いにより実行時間には大きな差が生じる。

本実験では、PyTorch では OOM (Out Of Memory) エラーが発生した行列サイズでも、CuPy では正常に処理が行えた。これは、両者の GPU メモリ管理の違いに起因していると考えられる。具体的には、PyTorch

は自動微分機構や高速化のためのキャッシュを保持するため、メモリを多く使いやすいが、CuPy は中間データを持たず、必要な分だけメモリを確保・解放するため、効率的なメモリ利用ができる。このため、CuPy の方がより大きな行列サイズでも OOM になりにくいという結果になったと考える。

### 3 行列の畳み込み

正方行列と  $3 \times 3$  行列の畳み込みの計算のサイズと実行時間の関係を調べ、python, numpy, pyTorch の比較を行う。行列のサイズは 1000 から 1000 ずつ増加させていく。行列のサイズが大きくなるにつれて、実行時間がどのように変化するかを調べる。

#### 3.1 プログラム

コード付録 B に、行列の畳み込みを計算する python プログラムを示す。

#### 3.2 実行結果

コード 2 と、図 2 に、行列の畳み込みの計算結果を示す。

コード 2 付録付録 B の実行結果

---

1	Size: 1000, Python: 0.9892s, NumPy: 0.0250s, PyTorch: 0.2096s
2	Size: 2000, Python: 3.9049s, NumPy: 0.0893s, PyTorch: 0.0019s
3	Size: 3000, Python: 8.3927s, NumPy: 0.1933s, PyTorch: 0.0115s
4	Size: 4000, Python: 14.9831s, NumPy: 0.3371s, PyTorch: 0.0207s
5	Size: 5000, Python: 23.3619s, NumPy: 0.5632s, PyTorch: 0.0432s
6	Size: 6000, Python: 33.6339s, NumPy: 0.7739s, PyTorch: 0.0475s
7	Size: 7000, Python: -----, NumPy: 1.0439s, PyTorch: 0.0668s
8	Size: 8000, Python: -----, NumPy: 1.3904s, PyTorch: 0.0864s
9	Size: 9000, Python: -----, NumPy: 1.7641s, PyTorch: 0.0133s
10	Size: 10000, Python: -----, NumPy: 2.6080s, PyTorch: 0.0171s
11	Size: 11000, Python: -----, NumPy: 7.7380s, PyTorch: 0.0209s
12	Size: 12000, Python: -----, NumPy: 9.2923s, PyTorch: 0.0260s
13	Size: 13000, Python: -----, NumPy: 11.3434s, PyTorch: 0.0278s
14	Size: 14000, Python: -----, NumPy: 6.0703s, PyTorch: 0.0320s
15	Size: 15000, Python: -----, NumPy: 14.5671s, PyTorch: 0.0380s
16	Size: 16000, Python: -----, NumPy: 15.5082s, PyTorch: 0.0430s
17	Size: 17000, Python: -----, NumPy: 6.8432s, PyTorch: 0.0448s
18	Size: 18000, Python: -----, NumPy: 7.9606s, PyTorch: 0.0465s
19	Size: 19000, Python: -----, NumPy: 8.6739s, PyTorch: 0.0479s
20	Size: 20000, Python: -----, NumPy: 9.1061s, PyTorch: 0.0511s

---

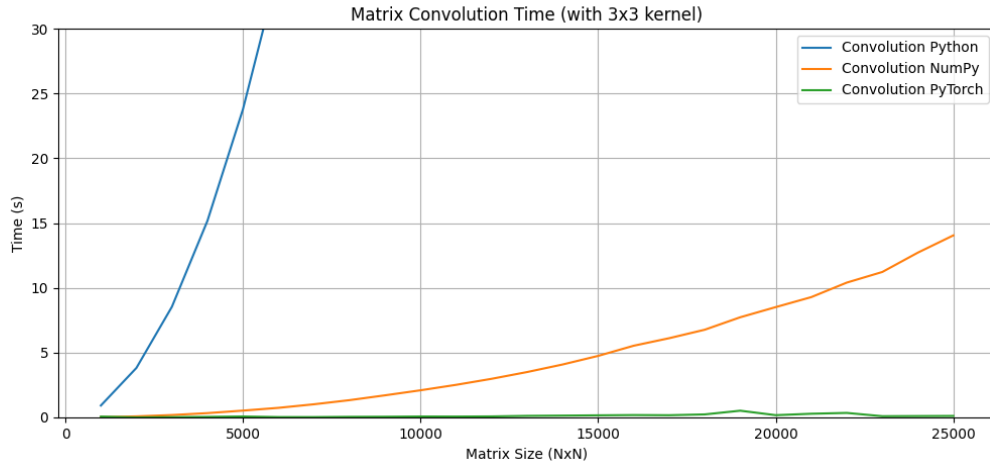


図2 付録付録 B の実行結果

### 3.3 考察

実行結果より、純粋 Python (ネストした for 文)、NumPy+SciPy (`‘scipy.signal.convolve2d’`)、CuPy (`‘cupyx.scipy.ndimage.convolve’`)、PyTorch (`‘torch.nn.functional.conv2d’`) のいずれも、理論上は  $O(N^2 k^2)$  の計算量を要するものの、実際の計測では大きく異なる挙動を示した。具体的には  $\text{Python} > \text{NumPy/SciPy} > \text{CuPy} \simeq \text{PyTorch}$  という順序で処理時間が短縮され、Python 実装ではループのオーバーヘッドや Python レイヤーでのメモリアクセスコストが大きいことが確認された。NumPy/SciPy は C 言語ベースのベクトル化処理により Python に比べて早いものの、あくまで CPU 上での実行であるため、並列化の限界やメモリ帯域幅の制約が性能を制限する要因となったと考える。一方、CuPy および PyTorch は GPU による並列計算を利用し、CPU ベースのライブラリに比べ桁違いに高速な畳み込みを実現した。

## 付録 A 2次元行列同士の積を求めるプログラム

コード 3 2次元行列同士の積を求めるプログラム

```
1 import time
2 import numpy as np
3 import cupy as cp
4 import torch
5 import matplotlib.pyplot as plt
6 import gc
7
8
9 def generate_matrix_python(size):
10     return [[(i + j) for j in range(size)] for i in range(size)]
11
12
13 def generate_matrix_np(size):
14     return np.array(
15         [[(i + j) for j in range(size)] for i in range(size)], dtype=np.float32
16     )
17
18
19 def generate_matrix_cp(size):
20     return cp.array(
21         [[(i + j) for j in range(size)] for i in range(size)], dtype=cp.float32
22     )
23
24
25 def generate_matrix_torch(size, device):
26     return torch.tensor(
27         [[(i + j) for j in range(size)] for i in range(size)],
28         dtype=torch.float32,
29         device=device,
30     )
31
32
33 def matrix_multiplication_python(u, v):
34     size = len(u)
35     result = [[0.0] * size for _ in range(size)]
36     for i in range(size):
37         for j in range(size):
38             for k in range(size):
39                 result[i][j] += u[i][k] * v[k][j]
40     return result
41
42
43 def matrix_multiplication_numpy(u, v):
44     return np.matmul(u, v)
45
46
47 def matrix_multiplication_cupy(u, v):
48     cp.cuda.Stream.null.synchronize()
49     result = cp.matmul(u, v)
50     cp.cuda.Stream.null.synchronize()
51     return result
52
53
54 def matrix_multiplication_torch(u, v):
55     torch.cuda.synchronize()
56     result = torch.matmul(u, v)
```

```

57     torch.cuda.synchronize()
58     return result
59
60
61 def benchmark():
62     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
63     step = 200
64     time_limit = 40
65
66     python_times = []
67     numpy_times = []
68     cupy_times = []
69     torch_times = []
70     multiple_sizes = []
71
72     i = 0
73     size = 0
74     flag_python = True
75     flag_numpy = True
76     flag_cupy = True
77     flag_torch = True
78     print("Starting benchmark...")
79     with open("multiply-result.txt", "w") as f:
80         while True:
81             print(
82                 f"flag_python: {flag_python}, flag_numpy: {flag_numpy}, flag_cupy: {
83                     flag_cupy}, flag_torch: {flag_torch}"
84             )
85             if not (flag_python or flag_numpy or flag_cupy or flag_torch):
86                 print("All methods exceeded the threshold.")
87                 break
88
89             size += step if flag_python else step * 5
90             print(f"Size: {size}")
91
92             print(f"Size: {size}", end="", \t", file=f)
93
94             # Python
95             if flag_python:
96                 u_py = generate_matrix_python(size)
97                 v_py = generate_matrix_python(size)
98                 start = time.time()
99                 matrix_multiplication_python(u_py, v_py)
100                 python_times.append(time.time() - start)
101                 print(f"Python: {python_times[-1]:.4f}s", end="", \t", file=f)
102                 del u_py, v_py
103                 if python_times[-1] > time_limit:
104                     flag_python = False
105             else:
106                 python_times.append(python_times[-1])
107                 print("Python: -----", end="", \t", file=f)
108
109             # NumPy
110             if flag_numpy:
111                 u_np = generate_matrix_np(size)
112                 v_np = generate_matrix_np(size)
113                 start = time.time()
114                 matrix_multiplication_numpy(u_np, v_np)
115                 numpy_times.append(time.time() - start)
116                 print(f"NumPy: {numpy_times[-1]:.4f}s", end="", \t", file=f)
117                 del u_np, v_np

```



```

117         if numpy_times[-1] > time_limit:
118             flag_numpy = False
119     else:
120         numpy_times.append(numpy_times[-1])
121         print("NumPy: -----", end=" \t", file=f)
122
123     # CuPy
124     if flag_cupy:
125         u_cp = generate_matrix_cp(size)
126         v_cp = generate_matrix_cp(size)
127         start = time.time()
128         matrix_multiplication_cupy(u_cp, v_cp)
129         cupy_times.append(time.time() - start)
130         print(f"Cupy: {cupy_times[-1]:.4f}s", end=" \t", file=f)
131         del u_cp, v_cp
132         if cupy_times[-1] > time_limit:
133             flag_cupy = False
134     else:
135         cupy_times.append(cupy_times[-1])
136         print("CuPy: -----", end=" \t", file=f)
137
138     # PyTorch
139     if flag_torch:
140         try:
141             u_th = generate_matrix_torch(size, device)
142             v_th = generate_matrix_torch(size, device)
143             start = time.time()
144             matrix_multiplication_torch(u_th, v_th)
145             torch_times.append(time.time() - start)
146             print(f"PyTorch: {torch_times[-1]:.4f}s", file=f)
147             if torch_times[-1] > time_limit:
148                 flag_torch = False
149         except Exception as e:
150             if "out of memory" in str(e):
151                 print("handling OOM error")
152                 print("PyTorch: OOM", file=f)
153                 u_th = None
154                 v_th = None
155                 torch_times.append(torch_times[-1])
156                 flag_torch = False
157             else:
158                 raise e
159         finally:
160             if u_th is not None and v_th is not None:
161                 del u_th, v_th
162                 torch.cuda.empty_cache()
163
164     else:
165         torch_times.append(torch_times[-1])
166         print("PyTorch: -----", file=f)
167
168     multiple_sizes.append(size)
169     gc.collect()
170     i += 1
171
172     # === Plotting ===
173     plt.figure(figsize=(12, 5))
174     plt.plot(multiple_sizes, python_times[: len(multiple_sizes)], label="Python")
175     plt.plot(multiple_sizes, numpy_times[: len(multiple_sizes)], label="NumPy")
176     plt.plot(multiple_sizes, cupy_times[: len(multiple_sizes)], label="Cupy")
177     plt.plot(multiple_sizes, torch_times[: len(multiple_sizes)], label="PyTorch")

```

```
178     plt.title("Matrix Multiplication Time")
179     plt.xlabel("Matrix Size (NxN)")
180     plt.ylabel("Time (s)")
181     plt.ylim(0, time_limit)
182     plt.legend()
183     plt.grid()
184     plt.savefig("matrix_multiplication_time.png")
185     plt.show()
186
187
188 if __name__ == "__main__":
189     benchmark()
```

---

## 付録 B 2次元行列の畳み込みを求めるプログラム

コード 4 2次元行列の畳み込みを求めるプログラム

```
1 import time
2 import numpy as np
3 import cupy as cp
4 import torch
5 import matplotlib.pyplot as plt
6 import gc
7 from scipy.signal import convolve2d
8 from cupyx.scipy.ndimage import convolve as cp_convolve
9
10
11 def generate_matrix(size):
12     return [[(i + j) % 2 for j in range(size)] for i in range(size)]
13
14
15 def generate_matrix_np(size):
16     return np.array(
17         [[(i + j) % 2 for j in range(size)] for i in range(size)], dtype=np.float32
18     )
19
20
21 def generate_matrix_cp(size):
22     return cp.array(
23         [[(i + j) % 2 for j in range(size)] for i in range(size)], dtype=cp.float32
24     )
25
26
27 def generate_matrix_torch(size, device):
28     return (
29         torch.tensor(
30             [[(i + j) % 2 for j in range(size)] for i in range(size)],
31             dtype=torch.float32,
32             device=device,
33         )
34         .unsqueeze(0)
35         .unsqueeze(0)
36     )
37
38
39 def convolution_python(img, kernel):
40     h, w = len(img), len(img[0])
41     kh, kw = len(kernel), len(kernel[0])
42     output = [[0.0 for _ in range(w - kw + 1)] for _ in range(h - kh + 1)]
43
44     for i in range(h - kh + 1):
45         for j in range(w - kw + 1):
46             for ki in range(kh):
47                 for kj in range(kw):
48                     output[i][j] += img[i + ki][j + kj] * kernel[ki][kj]
49     return output
50
51
52 def convolution_numpy(img, kernel):
53     return convolve2d(img, kernel, mode="valid")
54
55
56 def convolution_cupy(img, kernel):
```

```

57     cp.cuda.Stream.null.synchronize()
58     result = cp_convolve(img, kernel, mode="constant", cval=0.0)
59     cp.cuda.Stream.null.synchronize()
60     return result
61
62
63 def convolution_torch(img, kernel):
64     result = torch.nn.functional.conv2d(img, kernel, padding=0)
65     torch.cuda.synchronize()
66     return result
67
68
69 def benchmark():
70     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
71     step = 1000
72     time_limit = 30
73     max_size = 25000
74
75     kernel_py = [[1 / 9.0] * 3 for _ in range(3)]
76     kernel_np = np.ones((3, 3)) / 9.0
77     kernel_cp = cp.ones((3, 3), dtype=cp.float32) / 9.0
78     kernel_torch = torch.tensor(kernel_np).unsqueeze(0).unsqueeze(0).to(device).float()
79
80     python_times = []
81     numpy_times = []
82     cupy_times = []
83     torch_times = []
84     conv_sizes = []
85
86     size = 0
87     flag_python = True
88     flag_numpy = True
89     flag_cupy = True
90     flag_torch = True
91
92     print("Starting convolution benchmark...")
93
94     with open("conv-result.txt", "w") as f:
95         while True:
96             if not (flag_python or flag_numpy or flag_cupy or flag_torch):
97                 print("All methods exceeded the threshold.", file=f)
98                 break
99             if size >= max_size:
100                 print("Maximum size reached.", file=f)
101                 break
102
103             size += step
104             conv_sizes.append(size)
105             print(f"Size: {size}", end="\t", file=f)
106
107             # Python
108             if flag_python:
109                 u_py = generate_matrix(size)
110                 start = time.time()
111                 convolution_python(u_py, kernel_py)
112                 python_times.append(time.time() - start)
113                 print(f"Python: {python_times[-1]:.4f}s", end="\t", file=f)
114                 del u_py
115                 if python_times[-1] > time_limit:
116                     flag_python = False
117             else:

```

```

118         python_times.append(python_times[-1])
119         print("Python: -----", end="", \t", file=f)
120
121     # NumPy
122     if flag_numpy:
123         u_np = generate_matrix_np(size)
124         start = time.time()
125         convolution_numpy(u_np, kernel_np)
126         numpy_times.append(time.time() - start)
127         print(f"NumPy: {numpy_times[-1]:.4f}s", end="", \t", file=f)
128         del u_np
129         if numpy_times[-1] > time_limit:
130             flag_numpy = False
131     else:
132         numpy_times.append(numpy_times[-1])
133         print("NumPy: -----", end="", \t", file=f)
134
135     # CuPy
136     if flag_cupy:
137         try:
138             u_cp = generate_matrix_cp(size)
139             start = time.time()
140             convolution_cupy(u_cp, kernel_cp)
141             cupy_times.append(time.time() - start)
142             print(f"CuPy: {cupy_times[-1]:.4f}s", end="", \t", file=f)
143             del u_cp
144             if cupy_times[-1] > time_limit:
145                 flag_cupy = False
146         except Exception as e:
147             print("CuPy: Error", file=f)
148             cupy_times.append(cupy_times[-1] if cupy_times else 0)
149             flag_cupy = False
150     else:
151         cupy_times.append(cupy_times[-1])
152         print("CuPy: -----", end="", \t", file=f)
153
154     # PyTorch
155     if flag_torch:
156         try:
157             u_th = generate_matrix_torch(size, device)
158             start = time.time()
159             convolution_torch(u_th, kernel_torch)
160             torch_times.append(time.time() - start)
161             print(f"PyTorch: {torch_times[-1]:.4f}s", file=f)
162             if torch_times[-1] > time_limit:
163                 flag_torch = False
164         except Exception as e:
165             print("PyTorch: Error", file=f)
166             torch_times.append(torch_times[-1] if torch_times else 0)
167             flag_torch = False
168         finally:
169             if "u_th" in locals():
170                 del u_th
171             torch.cuda.empty_cache()
172     else:
173         torch_times.append(torch_times[-1])
174         print("PyTorch: -----", file=f)
175
176     gc.collect()
177
178     # === Plotting ===

```

```
179     plt.figure(figsize=(12, 5))
180     plt.plot(conv_sizes, python_times[: len(conv_sizes)], label="Python")
181     plt.plot(conv_sizes, numpy_times[: len(conv_sizes)], label="NumPy")
182     plt.plot(conv_sizes, cupy_times[: len(conv_sizes)], label="CuPy")
183     plt.plot(conv_sizes, torch_times[: len(conv_sizes)], label="PyTorch")
184     plt.title("Matrix Convolution Time (3x3 kernel)")
185     plt.xlabel("Matrix Size (NxN)")
186     plt.ylabel("Time (s)")
187     plt.ylim(0, time_limit)
188     plt.legend()
189     plt.grid()
190     plt.savefig("matrix_convolution_time.png")
191     plt.show()
192
193
194 if __name__ == "__main__":
195     benchmark()
```

---