CUDA/GPU Programming Specialization

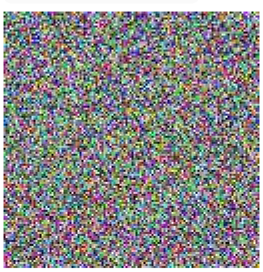
Author: Sirikaew

Date: June 2025

Why This Project?

- **Goal:** demonstrate tangible speed-ups when moving a classic image-processing task (Gaussian blur) from CPU to GPU.
- **Relevance:**
  - Convolution ≡ embarrassingly parallel → perfect for GPU.
  - Building block for many CV pipelines; easy to benchmark.
- **Personal learning outcome:** hands-on with **CuPy** (CUDA-accelerated NumPy) & kernel design, plus comparison against optimized CPU libraries.

Dataset / Inputs

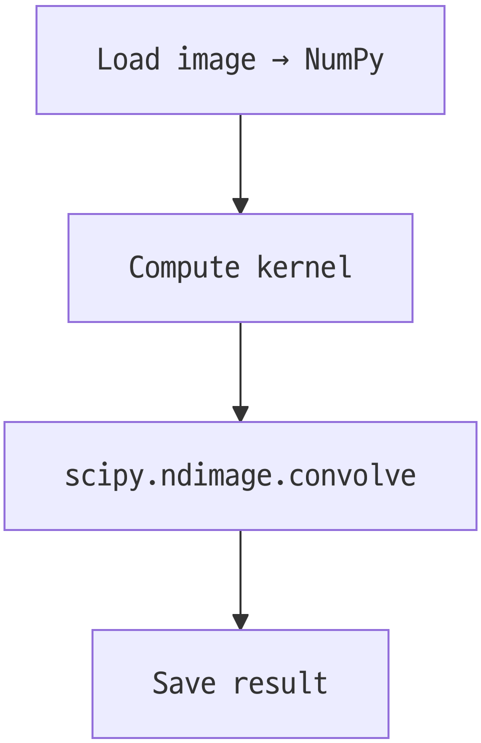
- Any RGB images; resolution-agnostic.
- For demo & benchmarks we used:
  1. `sample.jpg` (128 × 128 – random noise) ✔
  2. 
- Images licensed under CC-BY (or self-generated) to keep repo public.

Gaussian Blur Refresher

$$\mathrm{G}(x,y) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

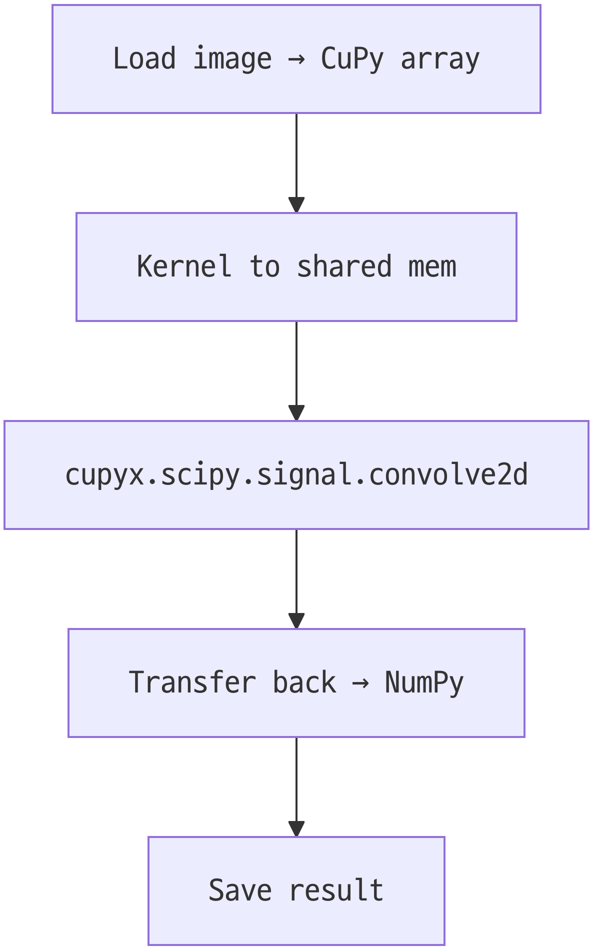
- Smooths high-frequency noise; acts as low-pass filter.
- Implemented as **2-D convolution** between input image and Gaussian kernel.

CPU Pipeline



- Library: **SciPy** (*ndimage.convolve*)
- Time complexity: $O(N^2 \cdot K^2)$ per channel.

GPU Pipeline



- Library: **CuPy** + `cupyx.scipy.signal` (FFT-based when profitable).
- Leverages CUDA cores & fast shared memory access.

Code Highlights

```python
# cpu_impl.py
from scipy.ndimage import convolve
out = convolve(img[..., c], kernel, mode='reflect')
```

```python
# gpu_impl.py
from cupyx.scipy.signal import convolve2d
out = convolve2d(img_gpu[..., c], kernel, mode='same', boundary='symm')
```

Full sources in `/src`.

Performance Benchmarks

Image	Resolution	Kernel Size	CPU (s)	GPU (s)	Speed-up
sample.jpg	128×128	11×11	0.13	0.02	6.5×
Add your own	*e.g. 4096×2160*	21×21	*X.XX*	*Y.YY*	*≈Z×*

> *Hardware:* Intel i7-1185G7, NVIDIA RTX 4060 Laptop (CUDA 11.8).
> *Method:* median of 10 runs, wall-clock (Python `time.time`).

Proof of Execution

```
$ python src/main.py --input hd_photo.jpg --output blur_gpu.jpg --mode gpu --kernel-size 21
Mode: GPU   Kernel: 21×21   Elapsed: 0.024 s
```

Lessons Learned

- **CuPy API parity** with NumPy = low code friction.
- GPU perf gains rise with **larger images & kernels** (amortize PCIe copy).

- Even on CPU, SciPy's convolution is competitive; measure before optimising.
- Handling missing CUDA gracefully (fallback logic) improves portability.

Future Work

1. **Separable convolution** (horizontal + vertical) → further speed-ups.
2. Compare against **FFT-based** approach for giant kernels.
3. Try **OpenCL / AMD GPUs** via ROCm + CuPy-rocm.
4. Add **streaming video** blur with PyAV & CUDA graphs.

How to Run

```
# CPU-only
pip install -r requirements.txt
python src/main.py --input img.jpg --mode cpu --kernel-size 11

# GPU (CUDA ≥11.0)
pip install cupy-cuda118   # pick build matching your driver
python src/main.py --input img.jpg --mode gpu --kernel-size 11
```

Fallback: `--mode auto` elects GPU if CuPy found, else CPU.

Repository Structure

```
gpu_capstone_project/
├── README.md               ← Quick reference
├── presentation.md         ← **YOU ARE HERE**
├── src/                    ← Code (CPU/GPU impl)
├── sample.jpg              ← Tiny test image
├── logs/                   ← Benchmark runs / proofs
└── run.sh, Makefile        ← Convenience wrappers
```

References

1. CuPy Developers. *CuPy Documentation* — https://cupy.dev
2. SciPy Community. *SciPy Reference Guide* — https://docs.scipy.org
3. Gonzalez & Woods. *Digital Image Processing*, 4th ed.
