Instituto Politécnico Nacional

“Escuela Superior de Cómputo”

Alumno:

- Ignacio Cortés Atzin Maxela

Unidad de aprendizaje: Cómputo paralelo

Profesor: Luis Alberto Ibáñez Zamora

Grupo: 6BM2

“Tarea: Tabla Comparativa de Librerías CUDA para Python”

Librería	Descripción / Enfoque principal	Nivel (abstracción)	Facilidad de uso	¿Necesita escribir kernels CUDA?	¿Soporta arrays tipo NumPy?	Casos de uso común	Soporte para Deep Learning / ML
PyCUDA	Acceso directo a CUDA desde Python	Bajo	Media	Sí	Limitado (GPUArray)	Kernels personalizados, control fino	No
CuPy	Reemplazo GPU-compatible de NumPy	Alto	Alta	No	Sí	Álgebra lineal, FFT, redes neuronales	Sí, compatible con Chainer
Numba	Compilador JIT con soporte CUDA	Medio	Alta	Opcional	Parcial	Aceleración de funciones Python. Funciones matemáticas, kernels simples, aceleración de bucles	Parcial (via sklearn, RAPIDS)
cuDF	DataFrames acelerados con GPU (similar a pandas)	Alto	Alta	No	No (usa DataFrame)	Ciencia de datos, análisis tabular, Análisis de datos, procesamiento de grandes datasets	Parcial, integración con RAPIDS
PyTorch	Framework para deep learning con soporte CUDA nativo	Alto	Alta	No	Sí (Tensors similares)	Redes neuronales, aprendizaje profundo, Deep learning, redes neuronales, investigación en IA	Sí

Análisis por Librería

1. PyCUDA

Es una interfaz directa a la API de CUDA, ideal para quienes necesitan control total sobre los recursos de la GPU. Permite definir kernels en lenguaje C/CUDA, compilarlos dinámicamente y ejecutarlos desde Python. Es muy flexible pero requiere un buen dominio de la arquitectura CUDA.

2. CuPy

Ofrece una sintaxis casi idéntica a NumPy, lo que facilita su adopción sin tener que aprender CUDA. Internamente utiliza CUDA para realizar cálculos en GPU, pero el usuario no necesita conocerlo en detalle. Es especialmente útil para cálculos matriciales y científicos.

3. Numba

Proporciona compilación Just-In-Time (JIT) y soporte para escribir funciones CUDA en Python. Es ideal para usuarios que quieren mejorar el rendimiento de bucles y operaciones numéricas sin salir de Python. Tiene soporte limitado para estructuras complejas pero ofrece gran flexibilidad.

4. cuDF

Forma parte del ecosistema RAPIDS de NVIDIA, enfocado en análisis de datos en GPU. Se usa principalmente para procesamiento de grandes datasets en formato tabular (similar a Pandas). Está construido sobre CUDA y ofrece alto rendimiento sin requerir conocimientos técnicos de GPU.

5. PyTorch

Es un framework de aprendizaje profundo con soporte nativo de CUDA. Los tensores pueden ser trasladados a la GPU automáticamente, permitiendo entrenamiento y predicción acelerados. Aunque no se programa CUDA directamente, está profundamente integrado con CUDA bajo el capó.

Conclusión Personal

La elección de una librería dependerá del tipo de problema, nivel de control deseado y familiaridad con CUDA. Si se busca máxima flexibilidad y control sobre los hilos, memoria y lanzamiento de kernels, PyCUDA es una excelente opción, aunque su curva de aprendizaje es alta. Por otro lado, si se prioriza la simplicidad y compatibilidad con el ecosistema científico de Python, CuPy y Numba ofrecen una transición natural hacia el cómputo en GPU sin sacrificar demasiado rendimiento.

Para tareas de ciencia de datos, cuDF destaca como alternativa eficiente y fácil de usar, mientras que PyTorch es insustituible cuando se trata de deep learning. En conjunto, estas herramientas representan un puente sólido entre la simplicidad de Python y la potencia bruta de las GPU, democratizando el acceso al cómputo de alto rendimiento.