

Programación paralela en Python

XI Semana de la Informática

Carlos García Sánchez

UCM

5 de febrero de 2025

- “Data Parallel C++ Programming Accelerated Systems Using C++ and SYCL”, James Reinders

Outline

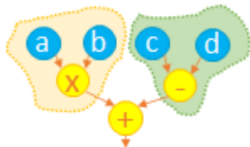
1 Data Parallel con dnp

Python

- Lenguaje de programación destacado por **productividad**
- ... pero con aplicaciones pesadas rendimiento puede verse comprometido
 - En la actualidad existen grandes esfuerzos por parte de la comunidad para optimizar paquetes empleados para “data-science”
 - ... incluso explotación de los **niveles de paralelismo en un computador moderno**

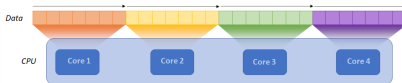
Niveles de paralelismo

- **ILP**: a nivel de instrucción si existen instrucciones independientes
 - En ej. instrucción '*' puede procesarse en paralelo con '-'

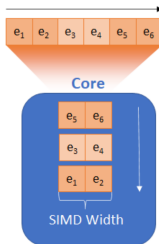


Niveles de paralelismo

- **Multiples cores:** multiples unidades de ejecución permiten procesar datos concurrentemente

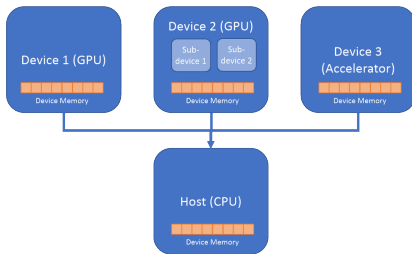


- **SIMD:** procesamiento en pequeños vectores
 - En ej. un SIMD=2 tardará la mitad



Computación heterogénea

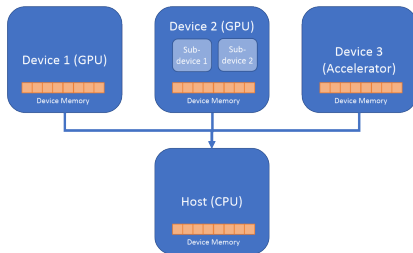
- Python es un lenguaje interpretado, normalmente el código se ejecutará en CPU
 - ... aunque conceptos de descargar código en “acelerador” o en los *offload devices* es útil para mejorar tiempos de ejecución



Computación heterogénea

■ Esquema host-device

- **Host** es el ejecutor del interprete de Python. Descarga trabajo en dispositivos
- **Device** (o varios dispositivos: 2 GPUs y un acelerador) puede ejecutar cargas de trabajo o *kernels*



Computación heterogénea

- [Data Parallel Extensions for Python](#) ofrece un modelo de programación donde el script de python permite la descarga de regiones paralelas en el/los dispositivo(s)
 - Se puede ejecutar multiples regiones paralelas y por tanto múltiples *offload kernels*
- Kernels son pre-compilados como librería como **dpnp** y alternativamente codificados en lenguajes heterogéneos como [OpenCL](#) o [DPC++](#)
- [Data Parallel Extensions for Python](#) permite la escritura directa en Python usando el compilador [Numba](#) con *numba-dpex* o la [extensión del Data Parallel para Numba](#)

CPU vs GPU

■ CPU

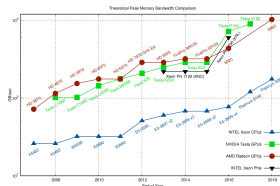
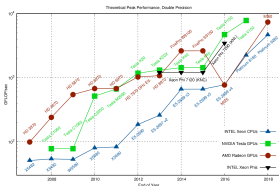
- Propósito general
- Bueno para el procesamiento en serie
- Excelente para el paralelismo de tareas
- Baja latencia por subproceso
- Caché y control dedicados de área grande

■ GPU

- Altamente especializado para el paralelismo
- Ideal para paralelismo de datos
- alto rendimiento
- Cientos de unidades de ejecución de punto flotante

CPU vs GPU

- GPU vs CPU (rendimiento)
 - GPU vista como co-procesadores de la CPU (**alto paralelismo**)
 - Descargan cómputo lanzado en *kernel* (lanzado asíncronamente)
 - CPU y GPU puede trabajar concurrentemente



Motivación

- Desafíos: muchos modelos de programación
 - Más abstracción vs Más Rendimiento

+++ Abstraction					+++ Performance
python	C/C++ Fortran	OpenMP (Cores)	OpenMP target OpenACC	OpenCL CUDA	Vector Intrs. GPU Intrinsics

Retos

- 1 (Variedad): Muchos lenguajes con sus toolchains, versión a mantener e integrar
- 2 (Performance): Desarrollo de app con alto rendimiento habitualmente conlleva desarrolladores especializados
- 3 (Porting): Algunas abstracciones ofrecen soluciones para alto-rendimiento en diferentes arquitecturas

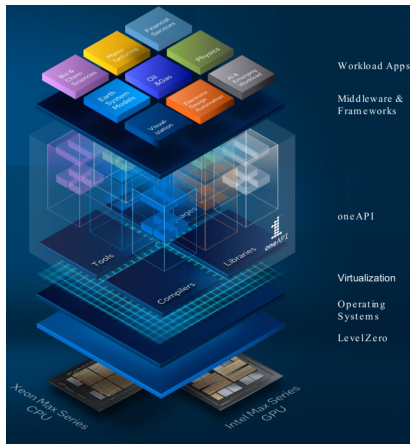
Algunas comparaciones

■ Algunos modelos de programación para aceleradores

	CUDA	OpenACC	OpenMP (5.0)	SYCL
Language	C/C++	C/C++ Fortran	C/C++ Fortran	C/C++
Prog. Style		pragmas	pragmas	C++11 lambdas
Parallelism	SIMT	SIMD, Fork/join CUDA	SPMD, SIMD Tasks, Fork/join, CUDA	OpenCL
Licensing	Proprietary	Few comp.	Open-source	Open-source
Abstraction	Low	High	High	Medium

oneAPI

- Modelo de programación unificado: diversas arquitecturas
- Lenguaje y bibliotecas optimizados
- Rendimiento equivalente lenguaje nativo de alto nivel
- Basado en estándares de la industria y especificaciones abiertas
- Compatible con los modelos de programación HPC

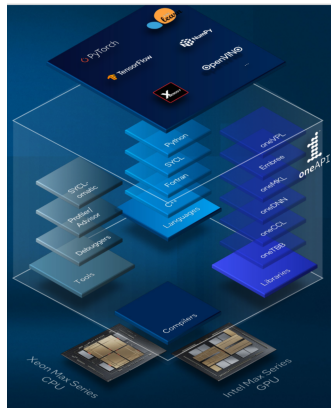


oneAPI

- Un lenguaje basado en estándares: C++ y SYCL
- Potentes API para acelerar funciones de dominio específico







Soluciones a proveedor único

- Estándar abierto para promover el apoyo de la comunidad y la industria
- Permite la reutilización de código en diferentes arquitecturas y proveedores



oneAPI

- Un conjunto completo de herramientas de desarrollo testeadas desde CPU a XPU
- Disponible para su instalación en **Intel® oneAPI Toolkits**

Intel® oneAPI Base Toolkit	 <p>A core set of high-performance libraries and tools for building C++, SYCL, C/OpenMP, and Python applications</p>		
Add-on Domain-specific Toolkits	 <p>For HPC developers</p> <p>Intel® oneAPI Tools for HPC Deliver fast Fortran, OpenMP & MPI applications that scale</p>	 <p>For Edge & IoT developers</p> <p>Intel® oneAPI Tools for IoT Build efficient, reliable solutions that run at network's edge</p>	 <p>For visual creators, scientists, and engineers</p> <p>Intel® oneAPI Rendering Toolkit Create performant, high-fidelity visualization applications</p>
	 <p>For AI developers and data scientists</p> <p>Intel® AI Analytics Toolkit Accelerate machine learning & data science pipelines end-to-end with optimized DL & ML frameworks & high-performing Python libraries</p>		
	 <p>For deep learning inference developers</p> <p>Intel® OpenVINO™ toolkit Deploy high performance inference & applications from edge to cloud</p>		
Toolkits powered by oneAPI			
Download at intel.com/oneAPI Or run tools in the Intel® Developer Cloud			

AI Tools Toolkit

- Acelera el flujo de trabajo desde un extremo al otro para aplicaciones IA y analítica de datos mediante librerías optimizadas para arquitecturas Intel
- ¿Para quién es interesante?
 - Desde científicos de datos, investigadores en IA, desarrolladores de aplicaciones IA y ML...
- Beneficios
 - Rendimiento en aplicaicones de DL desde el entrenamiento e inferencia: frameworks optimizados para archs Intel
 - Aceleraciones en aplicaciones de analítica de datos y ML intensivas en cómputo basadas en paquetes **python**

Numpy (vectorización)

- Reemplazando los bucles explícitos asociados a las operaciones entre vectores/arrays: SIMD
- En general operaciones entre arrays pueden lograrse entre 1 o 2 órdenes de magnitud de velocidad que las equivalentes *python puras*

Numpy (acelerada en oneAPI)

- Versión optimizada incluida en oneAPI
 - Última versión disponible en el [Intel® AI Analytics Toolkit \(AI Kit\)](#)

Python

■ Ventajas

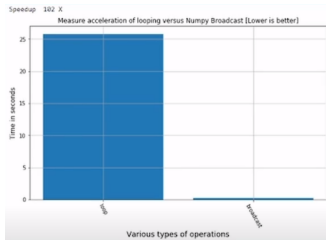
- Es rápido para desarrollar ideas
- Se puede codificar casi todo lo imaginable
- Desarrolla un proyecto rápidamente... se pueden encontrar ejemplos en cualquier sitio
- Fácil: sin tipado (dinámico) lo que hace la programación más sencilla
- Rápido: gran cantidad de librerías disponible y fáciles de instalar
- Aplicaciones de IA: sencillo de lograr portabilidad de los modelos desarrollados entre arquitecturas

Python

- ... pero es **lento**
 - Tareas de bajo nivel repetitivas
 - Bucles largos
 - Bucles anidados
- Pero hay formas de mitigar los problemas
 - Librerías optimizadas para las arquitecturas Intel
 - NumPy es un ejemplo
 - Otras librerías están también optimizadas

Vectorización

- La explotación de la vectorización **no es una teoría**
 - Altamente recomendable utilizar librerías optimizadas en Intel oneAPI como **NumPy**, **SciPy** y otras
 - Numpy (paralelismo inherente) permite explotar librerías optimizadas en oneAPI
- Ej: 100x de aceleración en Numpy *broadcasting* respecto al código descrito como bucle



Vectorización

- Python
 - Dinámicamente tipado
 - Hay que chequear el tipo de datos antes de operar...
- Incluso en operaciones tan “simples” como en enteros
 - Una clase o estructura puede contener: referencias (contadores) y otros valores

not vectorized

a		b
1	*	6
2	*	7
3	*	8
4	*	9
5	*	10

5 operations

vectorized

a		b
1	*	6
2		7
3		8
4		9
5	*	10

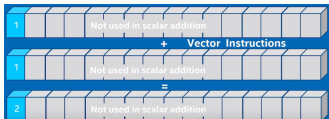
2 operations

Vectorización

- Las operaciones escalares (no vectorizadas) no son eficientes
 - Como llevar un autobús con un pasajero en una ciudad
 - ... trata de llenarlo

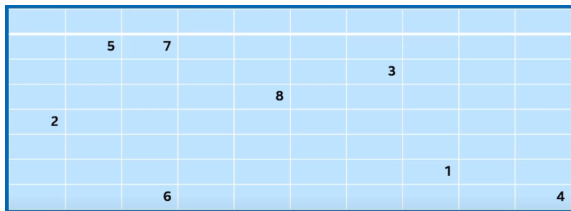
Terminal #1

```
>>> import numpy as np  
  
>>> a = np.array([1, 2, 3, 4, 5])  
>>> b = np.array([6, 7, 8, 9, 10])  
>>> a * b  
array([ 6, 14, 24, 36, 50])
```



Uso memoria ineficiente

- Una lista de enteros en python NO están contiguos en memoria
 - Ej: La lista [1, 2, 3, 4, 5, 6, 7, 8]
- El acceso a estos datos en un bucle es muy costoso (cientos de ciclos)



Uso memoria ineficiente

- Los accesos a memoria “random” matan el rendimiento
 - Las CPUs modernas tienen un jerarquía de memoria para explotar “localidad espacial de datos”
 - Una línea de cache puede contener 16 elementos consecutivos
 - ... accesos aleatorios conlleva **no explotar localidad espacial**
- Símil: *Cocinero cocina un huevo frito*
 - 1 Abre el cartón de huevos
 - 2 Saca el huevo
 - 3 Lo fríe
- Más eficiente si hace 12 huevos al mismo tiempo y los fríe en una sartén grande
 - Pero acceder a datos “desordenados” supone abrir varios cartones de huevos para poder freirlos

Uso memoria eficiente y SIMD

- El uso de memoria de forma eficiente. . .
- . . . accesos a datos consecutivos facilita la explotación SIMD
- El empaquetado de datos en el vector es inmediato y no conlleva hacer operaciones adicionales

Como puedo “reformular” el código a Numpy

- Numpy permite explotar la vectorización
 - Soporte de las funciones universales (ufuncs)
 - Indexación de datos evidente
 - Facilita broadcasting

Consejos

- Usar Numpy conlleva mejores patrones de acceso (explotación de jerarquía de memoria)
- Operaciones eficientes equivalentes a programar en “C”
- Reemplaza construcciones de bucles por construcciones con **ufuncs** incluso con condicionales:
 - Operaciones matemáticas: add, multiply, ... true_divide, power, log...
 - Operaciones trigonométricas: sin, cos... arctanh, ... degree, rad2deg...
 - Operaciones a nivel de bit: bitwise_and, ... left_shift,
 - Operaciones de comparación: greater, greater_equal, ... not_equal...
 - Operaciones de reducción

Paquete intel-numpy

- Disponible como paquete de [python](#)

Características

- NumPy optimizado con Intel® oneMKL para una multitarea, vectorización y gestión de memoria eficiente
- Proporciona objeto matriz N-dimensional, herramientas de integración (C/C++ y Fortran), funciones de álgebra lineal, transformadas de Fourier y números aleatorios. . .
- Distribuciones de NumPy en PyPI tienen licencia BSD

Hands-on

- Ejemplos que evidencian la mejora de rendimiento de Numpy en los [cuadernos de Jupyter ML Speedups](#)
 - 1 Cuaderno “01_Numpy_How_Fast_Are_Numpy_Ops.ipynb” que evalúa las ganancias de NumPy vs bucles
 - 2 Cuaderno “02_NumPy_UFUNCS.ipynb” presenta las ganancias de *ufuncs*
 - 3 Cuaderno “03_NumPy_Aggregations.ipynb” presenta mejoras en operaciones de map&reduce
 - 4 Cuaderno “04_NumPy_Broadcasting.ipynb” presenta mejoras en operaciones de expansión
 - 5 Cuaderno presenta ganancia en “selección” de valores

Hands-on

- Intel® Tiber™ AI Cloud disponible [en la URL](#)

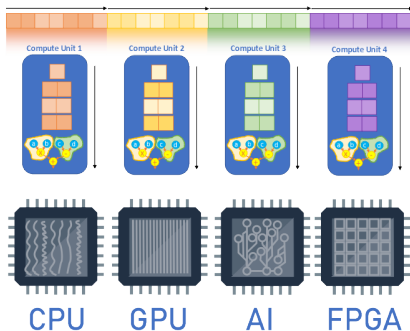


Instancias en Intel® Tiber™ Developer Cloud

- Core compute: Xeon 5th gen, Intel Core Ultra
- GPU: Intel Data Center GPU
- IA: Intel Gaudi HL

Data Parallel para Python

Data Parallel Extensions



Data Parallel para Python

- Las extensiones de [Data Parallel para Python](#) amplían las capacidades numéricas de Python
 - Más allá uso CPU: dispositivos de alto rendimiento como GPU
- Paquetes básicos:
 - **dpctl**: librería de control para seleccionar de dispositivos, la asignación de datos, datos tensoriales
 - **dpnp**: extensiones paralelas para Numpy
 - Implementa un subconjunto de Numpy
 - **numba_dpex**: extensiones del compilador Data Parallel para Numba
 - Extensión para el compilador Numba para programar dispositivos paralelos
- La implementación se basa en el estándar de [SYCL](#)

dpctl

- dpctl: contenedor ligero de Python sobre un subconjunto de la API de DPC++/SYCL
- Ej: selección de dispositivos

devices.py

```
import dpctl

d = dpctl.select_gpu_device()
d.print_device_info()
```

Terminal #1

```
Name           Intel(R) Graphics
Driver version  1.6.31294.120000
Vendor         Intel(R) Corporation
Filter string   level_zero:gpu:0
```

dpctl

- `dpctl.tensor` proporciona una descripción de un tensor N-dimensional
 - Tipos soportados `int{8,16,32,64}`, `float{16,32,64}`, `bool`, `complex{64,1128}`
- Creación de arrays (heredado de SYCL) soporta modelo USM
 - `to_device(device=target_device)`: mueve datos al dispositivo
 - `asarray()`: similar al anterior
 - `asnumpy(tensor)`: convierte un “`usm_ndarray`” en tipo numpy

dpctlExample.py

```
x_cpu = tensor.ones((2, 2), device="cpu")
x_gpu = tensor.ones((2, 2), device="gpu")
x_to_gpu = x_cpu.to_device("gpu")
x_np = numpy.ones((2,2))
x2_to_gpu = tensor.asarray(x_np, device="gpu")
```

dpctl

- `dpctl.tensor` proporciona una descripción de un tensor N-dimensional

dpctlExample.py

```
from dpctl import tensor
import numpy

x_cpu = tensor.ones((2, 2), device="cpu")
x_gpu = tensor.ones((2, 2), device="gpu")
x_to_gpu = x_cpu.to_device("gpu")
x_np = numpy.ones((2,2))
x2_to_gpu = tensor.asarray(x_np, device="gpu")

w = tensor.asarray([x_cpu, x_gpu, x_np, x_to_gpu, x2_to_gpu], device="cpu")
y = w + w + 1
print("Tensor y device:", y.device)
print("Tensor y shape:", y.shape)
y_np = tensor.asnumpy(y)
print("max: ", numpy.max(y_np))
```

dpctl

- `dpctl.tensor` proporciona una descripción de un tensor N-dimensional

Terminal #1

```
Tensor y device: Device(opencl:cpu:0)  
Tensor y shape: (5, 2, 2)  
max: 3.0
```

Datos escalares y arrays

- Tipos de datos básicos en Python y Numpy: float, int, complex
 - Representar datos escalares y se almacenan en el host
- Arrays se pueden almacenar en la memoria USM con `dpctl.tensor.usm_ndarray` y `dpnp.ndarray`
 - Recordar el concepto de [Compute-Follow-Data](#)

Extensión Data Parallel para NumPy-dpnp

- La librería dpnp (NumPy Drop-In Replacement for Intel(R) XPU) permite desarrollar aplicaciones en entorno NumPy en dispositivos paralelos
- Se puede partir de un script escrito en [NumPy](#)

```
# Original numpy code  
import numpy
```

```
X = np.array([1,2,3])  
Y = X * 4
```

Extensión Data Parallel para NumPy-dpnp

- Reemplazando un subconjunto de instancias a **numpy** ahora podemos ejecutar código en GPU

```
# Transformed numpy into dpnp
import dpnp as numpy
```

```
X = np.array([1,2,3])
```

```
Y = X * 4
```

```
print("Array X allocated on the device:", X.device)
```


Extensión Data Parallel para NumPy-dpnp

- `np.array` crea un array en el dispositivo SYCL por defecto, que puede ser una “gpu”
- De forma transparente para el desarrollador `x` se crea en el acelerador y se computa en el acelerador siguiendo el modelo “Compute-Follow-Data”

```
# Transformed numpy into dpnp
import dpnp as numpy
```

```
X = np.array([1,2,3])
Y = X * 4
print("Array X allocated on the device:", X.device)
```

Extensión Data Parallel para NumPy-dnp

- La creación de array tiene unos argumentos adicionales opcionales:
 - device: para indicar el dispositivo asociado: "host", "device" o "shared"
 - queue: para seleccionar la cola
 - usm_type: el tipo de memoria USM seleccionada

example2.py

```
import dnp as np

x = np.empty(3)
try:
    x = np.asarray([1, 2, 3], device="gpu")
except:
    print("GPU device is not available")

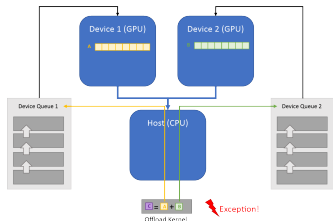
print("Array x allocated on the device:", x.device)

y = np.sum(x)
```

Extensión Data Parallel para NumPy-dpnp

- “Data Parallel Extensions for Python” requiere que los datos (tensores/arrays) estén alojados en la misma cola de ejecución

```
A=dp.array([1,2,3], device="gpu:0")  
B=dp.array([1,2,3], device="gpu:1")  
C=A+B # Error! Array are on different devices
```



Gestión de datos y cómputo

- Modelo **offload** soportado
 - API común en python para cómputo offload en sintaxis python
 - La descarga del kernel se produce donde los datos residen ('compute follows data')

```
X = dp.array([1,2,3])  
Y = X * 4
```

Executed on default device

```
X = dp.array([1,2,3], device="gpu:0")  
Y = X * 4
```

Executed on "gpu:0" device

```
X = dp.array([1,2,3], device="gpu:0")  
Y = dp.array([1,2,3], device="gpu:1")  
Z = X + Y
```

Error! Arrays are on different devices

Recursos disponibles

- Curso ML con oneAPI
- Iniciativa [oneAPI](#)
- Intel [IA Tools](#)
- Documentación de [Intel® Extension for Scikit-learn](#)
- Repo Intel® Extension for Scikit-learn en [GitHub](#)
- Documentación de [Intel® Extension for PyTorch](#)
- Repo IPEX en [GitHub](#)