Nombre del Alumno:	Diego Castro Elvira
Título de la tarea/práctica	Vectorización
Unidad de Aprendizaje	Cómputo Paralelo

Marco teórico

¿Qué es la vectorización en programación paralela?

La vectorización es la técnica de realizar operaciones en paralelo en elementos de datos en lugar de procesarlos de manera secuencial. En esencia, implica aplicar operaciones a todo un vector o arreglo de datos a la vez, en lugar de realizar operaciones elemento por elemento.

Beneficios de la vectorización en programación paralela:

1. Eficiencia en el uso del hardware: La vectorización permite aprovechar al máximo el hardware moderno al realizar múltiples operaciones en paralelo. Esto puede resultar en un aumento significativo en el rendimiento y la eficiencia de la ejecución de programas, especialmente cuando se trabaja con grandes conjuntos de datos.
2. Reducción de la sobrecarga de bucles: Al vectorizar operaciones en lugar de usar bucles explícitos, se reduce la cantidad de código necesario y se evita la sobrecarga asociada con la iteración en bucles. Esto puede conducir a un código más limpio y conciso, así como a un mejor rendimiento.
3. Optimizaciones a nivel de compilador: Muchos compiladores modernos son capaces de reconocer patrones de código que pueden beneficiarse de la vectorización y realizar automáticamente la transformación del código secuencial al código vectorizado. Esto significa que los programadores pueden concentrarse en escribir un código claro y legible, mientras que el compilador se encarga de realizar las optimizaciones necesarias para la ejecución eficiente en paralelo.

Técnicas de vectorización en programación paralela:

1. Instrucciones SIMD: Las instrucciones SIMD (Single Instruction, Multiple Data) son conjuntos de instrucciones en la CPU que permiten realizar operaciones en paralelo en varios elementos de datos al mismo tiempo. Estas instrucciones son especialmente útiles para realizar operaciones aritméticas y de manipulación de datos en grandes conjuntos de datos.
2. Uso de GPU: Las Unidades de Procesamiento Gráfico (GPU) están optimizadas para realizar operaciones en paralelo en grandes conjuntos de datos. Al utilizar la GPU en lugar de la CPU para ciertas tareas, se puede lograr un rendimiento aún

mayor al aprovechar las capacidades de procesamiento masivo en paralelo de la GPU.

Herramientas y librerías para vectorización en programación paralela:

1. NumPy: NumPy es una biblioteca de Python que proporciona soporte para matrices y operaciones matemáticas en ellas. NumPy utiliza internamente operaciones vectorizadas para mejorar el rendimiento.
2. TensorFlow y PyTorch: Estos son frameworks populares de aprendizaje automático que están optimizados para realizar operaciones en paralelo en grandes conjuntos de datos utilizando GPU.
3. Directivas de vectorización en compiladores: Algunos compiladores, como GCC y LLVM, proporcionan directivas específicas que permiten a los programadores indicar al compilador cómo vectorizar ciertas secciones de código.

Desarrollo

Crear dos programas donde se realicen las siguientes operaciones básicas:

1. Sumar
2. Restar
3. Multiplicar
4. Dividir

Estas operaciones se deben realizar a dos variables con un tamaño proporcionado y a cada una medir su tiempo de ejecución. Un programa debe seguir la programación secuencial y el otro seguir la programación paralela, siguiendo la técnica de vectorización. El objetivo principal es comparar los tiempos de ejecución de cada técnica y sacar conclusiones a partir de eso.

Resultado

Evaluando el programa de la programación secuencial (Apéndice A) y el programa de vectorización (Apéndice B) con 10 ejecuciones y un tamaño de vector de 100,000 obtenemos los siguientes resultados:

Operación	**Secuencial**		**Vectorización**	
	Media	*std*	*Media*	*std*
Suma	0.0471	0.008	0.0002	0.00009
Resta	0.0515	0.005	0.0003	0.00014
Multiplicación	0.0497	0.007	0.0002	0.00011
División	0.079	0.013	0.0004	0.00001
Total	0.228	0.028	0.0011	0.00039

Aplicando 100 ejecuciones y un tamaño de vector de 100,000 obtenemos:

Operación	Secuencial		Vectorización	
	Media	*std*	*Media*	*std*
Suma	0.0493	0.0128	0.0003	0.0001
Resta	0.0525	0.0134	0.0003	0.000009
Multiplicación	0.0541	0.0162	0.0002	0.0001
Division	0.0861	0.0202	0.0005	0.0001
Total	0.2421	0.0479	0.0015	0.0003

Conclusiones

En la programación secuencial, cada operación se realiza secuencialmente, es decir, una tras otra, esto significa que el procesador ejecuta una operación y luego pasa a la siguiente. Este enfoque puede ser eficiente para tareas pequeñas, pero a medida que aumenta el tamaño de los conjuntos de datos, el tiempo de ejecución aumenta de manera significativamente, como revisamos en las tablas. Comparándolo con la programación paralela con vectorización, las operaciones se realizan en paralelo en múltiples elementos de datos simultáneamente, como resultado, el tiempo de ejecución fue mucho menor, esto se nota más cuando aumentamos el tamaño de ejecuciones.

Apéndice

Apéndice A Programación secuencial

```
import numpy as np
from time import time

def secuential(n):

    a = np.random.randint(255, size=n)
    b = np.random.randint(255, size=n)

    suma_tiempo = []
    resta_tiempo = []
    multiplicacion_tiempo = []
```

```
    division_tiempo = []
    c = []

    # Operación de suma
    t = time()
    for i in range(len(a)):
        c.append(a[i] + b[i])
    suma_tiempo = time() - t

    # Operación de resta
    t = time()
    for i in range(len(a)):
        c.append(a[i] - b[i])
    resta_tiempo = time() - t

    # Operación de multiplicación
    t = time()
    for i in range(len(a)):
        c.append(a[i] * b[i])
    multiplicacion_tiempo = time() - t

    # Operación de división
    t = time()
    for i in range(len(a)):
        if b[i] != 0:
            c.append(a[i] / b[i])
    division_tiempo = time() - t

    tiempoTotal = suma_tiempo + resta_tiempo + multiplicacion_tiempo + division_tiempo

    return suma_tiempo, resta_tiempo, multiplicacion_tiempo, division_tiempo, tiempoTotal

# Se hacen 10 repeticiones
s = []
for i in range(10):
    s.append(secuential(100000))

means = np.mean(s, axis=0)
stds = np.std(s, axis=0)
print("Tiempo medio y desviación estándar de cada operación:\n")
print("Suma:", means[0], "s +/-", stds[0], "s")
print("Resta:", means[1], "s +/-", stds[1], "s")
print("Multiplicación:", means[2], "s +/-", stds[2], "s")
print("División:", means[3], "s +/-", stds[3], "s")
print("Tiempo Total:", means[4], "s +/-", stds[4], "s")
```

Apéndice B Programación paralela

```
# Se crea una función para realizar una rutina con vectorización
def vectorization(n):
    a=np.random.randint(255, size=n)
    b=np.random.randint(255, size=n)

    suma_tiempo = []
```

```
    resta_tiempo = []
    multiplicacion_tiempo = []
    division_tiempo = []

    # Operación de suma
    t=time()
    c=a+b
    suma_tiempo = time() - t

    # Operación de resta
    t=time()
    c=a-b
    resta_tiempo = time() - t

    # Operación de Multiplicación
    t=time()

    c=a*b
    multiplicacion_tiempo = time() - t

    # Operación de división
    t=time()

    c=a/b
    division_tiempo = time() - t

    sumaTotal= suma_tiempo + resta_tiempo + multiplicacion_tiempo + division_tiempo
    return suma_tiempo, resta_tiempo, multiplicacion_tiempo, division_tiempo, sumaTotal

# Se hacen 10 repeticiones
s=[]
for i in range(10):
    s.append(vectorization(100000))


means = np.mean(s, axis=0)
stds = np.std(s, axis=0)

print("Tiempo medio y desviación estándar de cada operación:\n")
print("Suma:", means[0], "s +/-", stds[0], "s")
print("Resta:", means[1], "s +/-", stds[1], "s")
print("Multiplicación:", means[2], "s +/-", stds[2], "s")
print("División:", means[3], "s +/-", stds[3], "s")
print("Tiempo Total:", means[4], "s +/-", stds[4], "s")
```