HIGH PERFORMANCE COMPUTING

Assignment-07

2303A51463

B-07

Serial Performance Baseline and Profiling

Assignment 1: SIMD Vector Addition

Prompt

Write a Python program that initializes two large arrays and performs element-wise vector addition using a normal Python loop and a NumPy vectorized operation. Measure and compare the execution time of both implementations and verify that the results are correct.

```
import numpy as np
import time

n = 10_000_000
A = np.random.rand(n)
B = np.random.rand(n)
# Scalar version
C = np.zeros(n)
start = time.time()
for i in range(n):
    C[i] = A[i] + B[i]
scalar_time = time.time() - start

# SIMD-like (NumPy)
start = time.time()
C2 = A + B
simd_time = time.time() - start

print("Scalar Time:", scalar_time)
print("SIMD Time:", simd_time)
print("Results Equal:", np.allclose(C, C2))
```

```
Scalar Time: 8.173792839050293
SIMD Time: 0.04368710517883301
Results Equal: True
```

Assignment 2: SIMD with Reduction Operation

Prompt

Write a Python program that initializes a large array with random values and computes the sum of its elements using a normal loop and a NumPy vectorized sum operation. Measure and compare the execution time of both methods and verify the correctness of the results.

```python
import numpy as np
import time

n = 10_000_000
A = np.random.rand(n)
# Scalar sum
start = time.time()
total = 0
for i in range(n):
    total += A[i]
scalar_time = time.time() - start
# SIMD reduction
start = time.time()
total2 = np.sum(A)
simd_time = time.time() - start

print("Scalar Sum:", total)
print("SIMD Sum:", total2)
print("Scalar Time:", scalar_time)
print("SIMD Time:", simd_time)
```

```
Scalar Sum: 5000870.413037799
SIMD Sum: 5000870.41303762
Scalar Time: 2.7056350708007812
SIMD Time: 0.009245157241821289
```

Assignment 3: Effect of Memory Alignment on SIMD

Prompt

Write a Python program that creates large arrays and performs SIMD-style vector addition using aligned and unaligned memory (using slicing for unaligned access). Measure and compare the execution time to analyze the effect of memory alignment on performance.

```python
import numpy as np
import time

n = 10_000_000
A = np.random.rand(n)
B = np.random.rand(n)
# Unaligned (slice creates offset)
A_unaligned = A[1:]
B_unaligned = B[1:]

# Aligned
start = time.time()
C1 = A + B
aligned_time = time.time() - start

# Unaligned
start = time.time()
C2 = A_unaligned + B_unaligned
unaligned_time = time.time() - start

print("Aligned Time:", aligned_time)
print("Unaligned Time:", unaligned_time)
```

```
Aligned Time: 0.032743215560913086
Unaligned Time: 0.03964567184448242
```

Assignment 4: Combining SIMD and OpenMP Parallel Loops

Prompt

Write a Python program that performs vector addition using a normal single-threaded approach and a NumPy vectorized approach. Measure and compare the execution time to analyze the benefits of data-level parallelism over scalar execution.

+ Code + Text

```
import numpy as np
import time

n = 10_000_000
A = np.random.rand(n)
B = np.random.rand(n)

# Single thread
start = time.time()
C1 = [A[i] + B[i] for i in range(n)]
scalar_time = time.time() - start

# SIMD (NumPy)
start = time.time()
C2 = A + B
simd_time = time.time() - start

print("Single Thread Time:", scalar_time)
print("Vectorized Time:", simd_time)
```

```
Single Thread Time: 3.6750800609588623
Vectorized Time: 0.03596615791320801
```

Assignment 5: Load Imbalance and SIMD Efficiency

Prompt

Write a Python program that performs computations inside a loop with conditional statements and measures execution time using a normal loop. Then rewrite the computation using NumPy vectorization (without explicit branching) and compare the performance to understand the impact of branch divergence on SIMD efficiency.

```
import numpy as np
import time
n = 10_000_000
A = np.random.rand(n)
B = np.zeros(n)
# Scalar with branch
start = time.time()
for i in range(n):
    if A[i] > 0.5:
        B[i] = A[i] * 2
    else:
        B[i] = A[i] / 2
scalar_time = time.time() - start
# SIMD-friendly
start = time.time()
B2 = np.where(A > 0.5, A * 2, A / 2)
simd_time = time.time() - start

print("Scalar Time:", scalar_time)
print("Vectorized Time:", simd_time)
```

```
Scalar Time: 6.781891345977783
Vectorized Time: 0.14166259765625
```