High Performance Computing

Sony Dodla

2303A51666

Batch-23

Assignment 3: Load Balancing with Irregular Workloads

Scenario

An image processing pipeline processes images with different resolutions, leading to uneven computation time.

Objective

Study load imbalance in parallel loops.

Tasks

1.Simulate variable workload per iteration.

2.Parallelize using prange.

3.Observe execution time variation.

4.Discuss limitations of scheduling in Python.

Learning Outcomes

- Load imbalance
- Comparison with OpenMP scheduling
- Practical HPC limitations in Python

Python code

```
import numpy as np

import time

from numba import njit, prange

np.random.seed(0)

n_images = 50

# Simulate different image resolutions (irregular workload)

resolutions = np.random.randint(50_000, 500_000, size=n_images)

# ----------------------------------------

# Simulate image processing workload
```

```
# ----------------------------------------
@njit
def process_image(resolution):
    """
    Artificially heavy computation proportional to image resolution.
    """
    acc = 0.0
    for i in range(resolution):
        acc += np.sqrt(i * 0.5)
    return acc
# ----------------------------------------
# Serial version
# ----------------------------------------
@njit
def serial_processing(resolutions):
    results = np.zeros(len(resolutions))
    for i in range(len(resolutions)):
        results[i] = process_image(resolutions[i])
    return results
# Serial timing
t0 = time.time()
serial_processing(resolutions)
t1 = time.time()
print("Number of images:", n_images)
print("Min resolution:", resolutions.min())
print("Max resolution:", resolutions.max())
print(f"Serial execution time   : {t1 - t0:.4f} s")
# ----------------------------------------
# Parallel version using prange
# ----------------------------------------
@njit(parallel=True)
```

```
def parallel_processing(resolutions):
    results = np.zeros(len(resolutions))
    for i in prange(len(resolutions)):
        results[i] = process_image(resolutions[i])
    return results
 # Parallel timing
t2 = time.time()
parallel_processing(resolutions)
t3 = time.time()
print(f"Parallel execution time : {t3 - t2:.4f} s")
```

Output:

```
Number of images: 50
Min resolution: 57877
Max resolution: 498242
Serial execution time   : 0.2397 s
Parallel execution time : 0.3970 s
```

C code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#define N_IMAGES 1000
#define MAX_THREADS 64
double process_image(int resolution)
{
    double acc = 0.0;
    for (int k = 0; k < resolution; k++)
        acc += sqrt(k * 0.5);
    return acc;
```

```
}
void run_schedule(int *resolution, omp_sched_t sched_type, const char *name)
{
    int i;
    double total = 0.0;
    int nthreads = omp_get_max_threads();
    int work[MAX_THREADS] = {0};
    double t_start[MAX_THREADS] = {0.0};
    double t_end[MAX_THREADS]   = {0.0};
    omp_set_schedule(sched_type, 0);
    double start = omp_get_wtime();
    #pragma omp parallel private(i)
    {
        int tid = omp_get_thread_num();
        t_start[tid] = omp_get_wtime();
        #pragma omp for schedule(runtime) reduction(+:total)
        for (i = 0; i < N_IMAGES; i++) {
            process_image(resolution[i]);
            work[tid]++;
        }
        t_end[tid] = omp_get_wtime();
    }
    double end = omp_get_wtime();
    printf("\n=== %s scheduling ===\n", name);
    printf("Total execution time: %f s\n", end - start);
    printf("Per-thread workload:\n");
    for (i = 0; i < nthreads; i++) {
        printf("Thread %2d | images: %4d | active time: %6.3f s\n",
               i, work[i], t_end[i] - t_start[i]);
    }
}
```

```
int main()
{
   int resolution[N_IMAGES];
   srand(0);
   for (int i = 0; i < N_IMAGES; i++)
      resolution[i] = 10000 + rand() % 200000;
   printf("Images  : %d\n", N_IMAGES);
   printf("Threads : %d\n", omp_get_max_threads());
   run_schedule(resolution, omp_sched_static,  "STATIC");
   run_schedule(resolution, omp_sched_dynamic, "DYNAMIC");
   run_schedule(resolution, omp_sched_guided,  "GUIDED");
   return 0;
}
```

Output:

```
Images  : 1000
Threads : 4
=== STATIC scheduling ===
Total execution time: 0.466385 s
Per-thread workload:
Thread  0 | images:  250 | active time:  0.454 s
Thread  1 | images:  250 | active time:  0.441 s
Thread  2 | images:  250 | active time:  0.451 s
Thread  3 | images:  250 | active time:  0.457 s
=== DYNAMIC scheduling ===
Total execution time: 0.427426 s
Per-thread workload:
Thread  0 | images:  191 | active time:  0.427 s
Thread  1 | images:  177 | active time:  0.423 s
Thread  2 | images:  320 | active time:  0.420 s
Thread  3 | images:  312 | active time:  0.427 s
=== GUIDED scheduling ===
```

Total execution time: 0.497476 s

Per-thread workload:

Thread 0 | images: 233 | active time: 0.497 s

Thread 1 | images: 305 | active time: 0.496 s

Thread 2 | images: 185 | active time: 0.496 s

Thread 3 | images: 277 | active time: 0.438 s