بسمه تعالی

عنوان: فاز دوم پروژه

استاد:

دکتر سپیده صفری

مسئول:

آقای معین اثنی عشری

نویسندگان:

امیررضا آذری 99101087

بزرگمهر ضیا 99100422

سیستم‌های بی‌درنگ

پاییز 1403

مقدمه:

در فاز اول، بخش آفلاین این زمان‌بندی همراه الگوریتم یادگیری تقویتی را انجام دادیم. بخش TDP از آن فاز ناقص مانده بود که در فاز دوم آن را تکمیل نمودیم. در این فاز وارد بخش آنلاین شدیم و تسک‌های نامتناوب نرم با زمان‌های ریلیز متفاوت وارد سامانه می‌شوند. الگوریتم یادگیری تقویتی ما باید به کمک زمان لختی، به نحوی زمان‌بندی را انجام بدهد تا تسک‌های سخت، موعد زمانی خود را رد نکنند و به نوعی فاجعه رخ ندهد. همچنین در این فاز به نمودارهای خواسته شده در صورت مسئله پروژه نیز پرداختیم و آن‌ها را با مقادیر مختلف تعداد هسته و میزان بهره‌وری آزمودیم. حال در این گزارش سعی می‌کنیم مجددا تمامی نکات کد را بیان نموده و چند خروجی را نیز به نمایش بگذاریم.

کتابخانه‌های مورد نیاز:

```
import copy
import random
import re
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
```

همان‌طور که در شکل و کد مشاهده می‌کنید، کتابخانه torch برای بخش یادگیری تقویتی باید اضافه بشود. همچنین کتابخانه‌های numpy و matplotlib برای نمایش خروجی ضروری می‌باشند.

الگوریتم UUNIFAST:

الگوریتم UUniFast برای تولید استفاده پذیری تسک‌ها استفاده می شود. هدف این الگوریتم توزیع یک مقدار کل استفاده پذیری بین تعداد مشخصی از تسک ها است، به گونه ای که مجموع استفاده پذیری‌ها برابر مقدار کل باشد.

جزئیات پیاده‌سازی:

- ورودی‌ها:

✓ `num_tasks` : تعداد تسک‌هایی که باید تولید شوند.
✓ `total_utilization` : مجموع استفاده‌پذیری که باید بین تسک‌ها تقسیم شود.

- منطق:

✓ باقی‌مانده استفاده‌پذیری (sum_u) به طور تکراری بین تسک‌ها تقسیم می‌شود.
✓ یک کسر تصادفی تعیین می‌کند چه مقدار از استفاده‌پذیری باقی‌مانده به تسک فعلی اختصاص داده شود.
✓ تسک نهایی تمام استفاده‌پذیری باقی‌مانده را دریافت می‌کند.

- خطوط کلیدی کد:

✓ `next_u = sum_u * (random.uniform(0, 1) ** (1 / (num_tasks - i)))`: تعیین تصادفی کسر بعدی استفاده‌پذیری.
✓ `utilizations.append(sum_u - next_u)`: اضافه کردن استفاده‌پذیری محاسبه‌شده برای تسک فعلی.

- خروجی:

✓ یک آرایه از بهره‌وری‌ها که مجموع آن برابر total_utilization است.

```python
def uunifast(num_tasks, total_utilization):
    utilizations = []
    sum_u = total_utilization

    for i in range(1, num_tasks):
        next_u = sum_u * (random.uniform(0, 1) ** (1 / (num_tasks - i)))
        utilizations.append(sum_u - next_u)
        sum_u = next_u

    utilizations.append(sum_u)
    return utilizations
```

تولید تسک‌ها:

تسک‌ها با استفاده از استفاده‌پذیری‌های تولیدشده توسط UUniFast تولید می‌شوند. هر تسک دارای ویژگی‌هایی مانند دوره، بدترین زمان اجرا(WCET) ، و ددلاین است.

جزئیات پیاده‌سازی:

- ورودی‌ها:

✓ num_tasks, total_utilization: به UUniFast ارسال می‌شود.
✓ min_period, max_period: محدوده‌ای برای دوره تسک‌ها تعریف می‌کند.

- منطق:

✓ دوره هر تسک به صورت تصادفی در محدوده انتخاب می‌شود.
✓ WCETبه صورت utilization * period محاسبه می‌شود.
✓ ددلاین‌ها برابر دوره‌ها تنظیم می‌شوند.

- خطوط کلیدی کد:

✓ انتخاب تصادفی دوره تسک: period = random.randint(min_period, max_period)

✓ محاسبه WCET: wcet = utilization * period

- خروجی:

یک لیست از تسک‌ها که هر تسک به صورت یک دیکشنری با ویژگی‌های زیر نمایش داده می‌شود:

✓ task_id: شناسه.

✓ utilization, period, wcet, deadline.

یکی از تفاوت‌های این بخش با فاز قبلی، تولید تسک‌های نامتناوب نرم می‌باشد. در این بخش، بهره‌وری را رندوم حساب کرده و در ادامه بدترین زمان اجرا را می‌یابیم. همچنین یک زمان رسیدن برای هر کدام تعیین می‌کنیم.

```
for i in range(num_aperiodic):
    utilization = random.uniform(0, 0.3)  # Random utilization for aperiodic task
    wcet = utilization * max_period  # Soft aperiodic task doesn't have a fixed period, use max_period as a base
    arrival_time = random.randint(0, max_arrival_time)  # Random arrival time
    deadline = random.randint(arrival_time + 1, max_arrival_time + 1)  # Random deadline after arrival

    aperiodic_task = {
        "task_id": tasks_len + i,
        "utilization": round(utilization, 4),
        "wcet": round(wcet / 10, 4),
        "arrival_time": arrival_time,
        "deadline": deadline,
        "type": "soft"
    }
    tasks.append(aperiodic_task)
```

همچنین تفاوت دیگر، اضافه شدن type به ویژگی تسک‌ها می‌باشد.

یادگیری تقویتی:

عامل RL یاد می‌گیرد که تسک‌ها را به کورها تخصیص دهد با استفاده از شبکه Q عمیق (DQN). محیط شبیه‌سازی اجراهای تسک و محاسبه پاداش بر اساس بهره‌وری انرژی و رعایت ددلاین است.

کلاس DVFSEnvironment

- هدف:
  - ✓ مدل‌سازی کورها و تسک‌ها.
  - ✓ شبیه‌سازی اجرای تسک و محاسبه معیارهایی مانند مصرف انرژی.
- متودها:
  - ✓ initialize_state: تنظیم اولیه کورها و صف تسک.
  - ✓ step: اجرای یک اقدام (تخصیص تسک به کور) و به‌روزرسانی وضعیت.
  - ✓ reset: بازنشانی محیط برای یک قسمت جدید.

حلقه آموزش

- ✓ در هر قسمت، عامل RL با محیط تعامل دارد.
- ✓ اقدامات بر اساس مقادیر Q انتخاب می‌شوند و شبکه با استفاده از معادله بلمن آموزش داده می‌شود.
- ✓ تخصیص نهایی تسک به کورها ذخیره می‌شود.

همچنین در این بخش، محاسبه زمان‌های لختی اضافه شده و همچنین شبکه DQN که اضافه بود از این فاز حذف شده است. در تابع step تغییرات زیادی اعمال نمودیم.

در این فاز، ولتاژ و فرکانس را بر اساس Arm Cortex-A7 ورودی داده و DVFS را به طور درست‌تری پیاده نمودیم. به طوری که زمان اجرای تسک‌ها بر اساس فرکانس آن‌ها تغییر می‌کند. همچنین توان را از فرمول آن محاسبه می‌نماییم. در بخش reward نیز تغییراتی را اعمال نموده‌ایم. اولین آن همان TDP است که اگر آن را رد کنیم، ریوارد منفی دریافت خواهیم کرد. همچنین برای تسک‌های نرم، زمان لختی را محاسبه کرده و اگر با

محاسبات انجام شده، تسک نرم آن موعد را رد نماید، یک ریوارد منفی با مقدار کمتر (زیرا نرم است و از دست دادن ددلاین فاجعه‌بار نخواهد بود) نسبت می‌دهیم.

```
frequency = self.frequencies[freq_id]
voltage = self.voltages[freq_id]

wcet = task["wcet"] / frequency
task["frequency"] = frequency
task["voltage"] = voltage
core["load"] += wcet / frequency
core["frequency"] = frequency

energy = frequency ** 2 * voltage ** 2 * wcet
self.energy_consumption += energy

reward = -energy
if self.energy_consumption > self.tdp:
    reward -= 1000
    self.energy_consumption -= energy
elif core["load"] > task["deadline"]:
    reward -= 100
else:
    task["wcet"] = wcet

if task["type"] == "soft":
    self.calculate_slack_time()
    if task["arrival_time"] > core["load"] or task["wcet"] > core["slack_time"]:
        reward -= 500
    else:
        core["slack_time"] -= task["wcet"]
```

بخش train:

در این بخش تغییرات زیادی مخصوصا برای محاسبه اطلاعات لازم برای خروجی نمودارها اعمال نمودیم.

```
powers = []
core_powers = {core: 0 for core in range(num_cores)}
all_powers = []
quality_of_services = {}
with open(f"results/scheduling_cores_{num_cores}_utilization_{total_utilization}.txt", "w") as file:
    for core_id, assigned_tasks in final_assignments.items():
        file.write(f"Core {core_id} (sorted by deadline):\n")
        current_time = 0  # Track the current time on each core
        total_power = 0
        for task in assigned_tasks:
            start_time = max(current_time, task["arrival_time"]) if task["type"] == "soft" else current_time
            finish_time = start_time + task["wcet"]
            if task["type"] == "soft":
                file.write(
                    f"  Task {task['task_id']} -> Type: {task['type']}, Arrival: {task['arrival_time']}, Start: {start_time:.2f}, Finish: {fin
            else:
                file.write(
                    f"  Task {task['task_id']} -> Type: {task['type']}, Start: {start_time:.2f}, Finish: {finish_time:.2f}, Deadline: {task['
```

توان هر تسک در هر زمان و همچنین خروجی ویژگی هر تسک بر اساس نوع آن را داریم. همچنین برای محاسبه میزان کیفیت خدمات، از فرمول آن استفاده نمودیم. این فرمول را از یکی از مقالات خانم دکتر صفری پیدا کرده‌ایم.

Optimization Goal: Maximize the QoS of the system defined by the average of the QoS of all tasks. Note that the probability of successful execution of each main task T_i is defined by P_i, and the probability of its failure is defined by 1-P_i.

$$\text{Maximize } QoS_{sys} = \frac{\sum_{i\in\Phi} P_i \times UF_i(f_{T_i}, D_i) + \sum_i (1-P_i) \times UF_i(f_{B_i}, D_i)}{n} \quad (8)$$

where UF_i is the utility function of task T_i that describes the utility value of the mentioned task as a function of its finishing time, and f_i is the finish time of T_i. In this paper, we have considered the linear function and can be rewritten as [14][34]:

$$UF_i = \begin{cases} 1 & f_i \le D_i \\ \dfrac{(D_i - f_i)}{D_i(x-1)} + 1 & D_i < f_i \le x \times D_i \\ 0 & f_i > x \times D_i \end{cases} \quad (9)$$

where x is a variable bigger than 1 in order to determine

پیاده‌سازی آن در این بخش کد موجود است:

```
if x_parameter * task['deadline'] >= finish_time > task['deadline']:
    quality_of_services[task['task_id']] = ((task['deadline'] - finish_time) / (
            task['deadline'] * (x_parameter - 1))) + 1
elif finish_time > x_parameter * task['deadline']:
    quality_of_services[task['task_id']] = 0
else:
    quality_of_services[task['task_id']] = 1

current_time = finish_time
core_utilizations[core_id] += task["utilization"]
total_power += task["frequency"] ** 2 * task["voltage"] ** 2 * task['wcet']

core_powers[core_id] += task['frequency'] ** 2 * task['voltage'] ** 2 * task['wcet']
all_powers.append(copy.deepcopy(core_powers))
```

همچنین توان مصرفی هسته‌ها نیز در این بخش محاسبه شده‌اند. باقی بخش کد بحث خروجی آن است که توضیح زیادی نمی‌طلبد اما کد هر یک همراه یک مثال خروجی را در این بخش می‌آوریم.

نمودار میزان کیفیت خدمات وظایف:

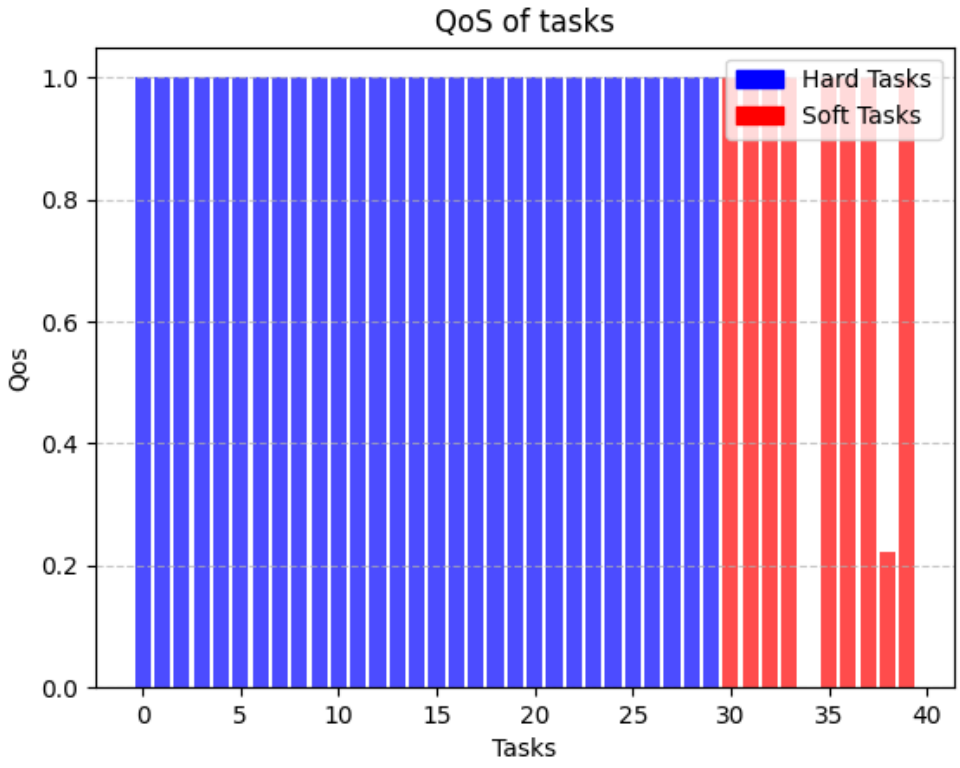
```
def plot_qos_tasks(num_aperiodic, num_cores, num_tasks, task_ids, total_utilization, values):
    x_labels = [f"T {i}" for i in task_ids]
    colors = ['r' if task_id >= (num_tasks - num_aperiodic) else 'b' for task_id in task_ids]
    plt.bar(task_ids, values, color=colors, alpha=0.7)
    blue_patch = mpatches.Patch(color='b', label='Hard Tasks')
    red_patch = mpatches.Patch(color='r', label='Soft Tasks')
    plt.legend(handles=[blue_patch, red_patch], loc='upper right')
    plt.xlabel("Tasks")
    plt.ylabel("Qos")
    plt.title("QoS of tasks")
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.savefig(f"results/output1_core_{num_cores}_utilization_{total_utilization}.png")
    plt.close()
```

این تابع بعد محاسبه شدن میزان کیفیت بر اساس فرمولی که بالاتر اشاره شد، صدا زده می‌شود.

```
            if x_parameter * task['deadline'] >= finish_time > task['deadline']:
                quality_of_services[task['task_id']] = ((task['deadline'] - finish_time) / (
                        task['deadline'] * (x_parameter - 1))) + 1
            elif finish_time > x_parameter * task['deadline']:
                quality_of_services[task['task_id']] = 0
            else:
                quality_of_services[task['task_id']] = 1

            current_time = finish_time
            core_utilizations[core_id] += task["utilization"]
            total_power += task["frequency"] ** 2 * task["voltage"] ** 2 * task['wcet']

            core_powers[core_id] += task['frequency'] ** 2 * task['voltage'] ** 2 * task['wcet']
            all_powers.append(copy.deepcopy(core_powers))

        powers.append(total_power)

task_ids = list(quality_of_services.keys())
values = list(quality_of_services.values())
mean_qos = np.mean(values)

plot_qos_tasks(num_aperiodic, num_cores, num_tasks, task_ids, total_utilization, values)
```

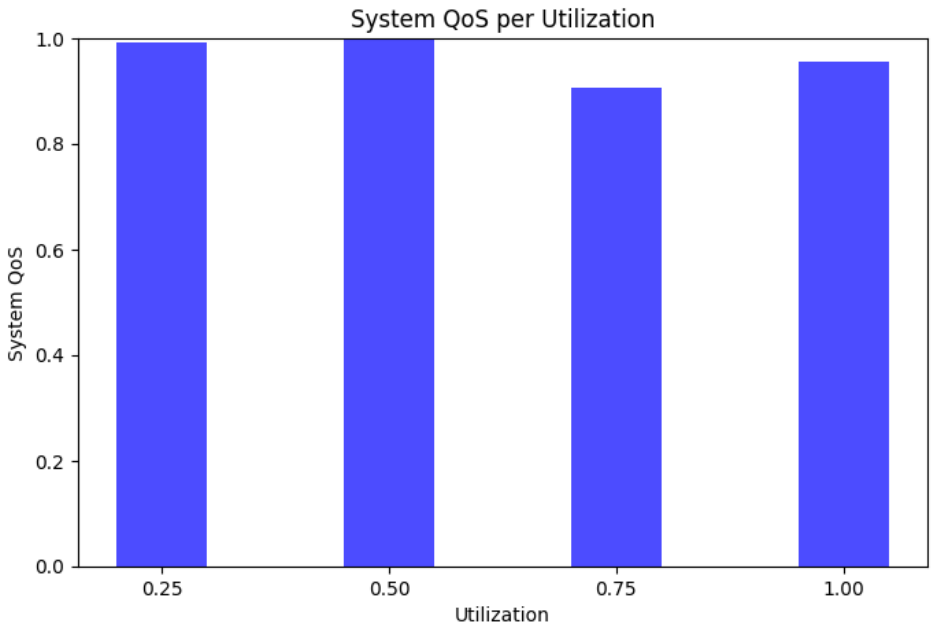
نمونه خروجی:

همانطور که مشاهده می‌کنید یک سری تسک‌های نرم مقدار صفر را گرفتند زیرا از ددلاین خود مقدار بیشتری عبور کرده‌اند. اما تسک‌هایی که خیلی فاصله نگرفتند، مقداری بین 0 تا 1 دارند.



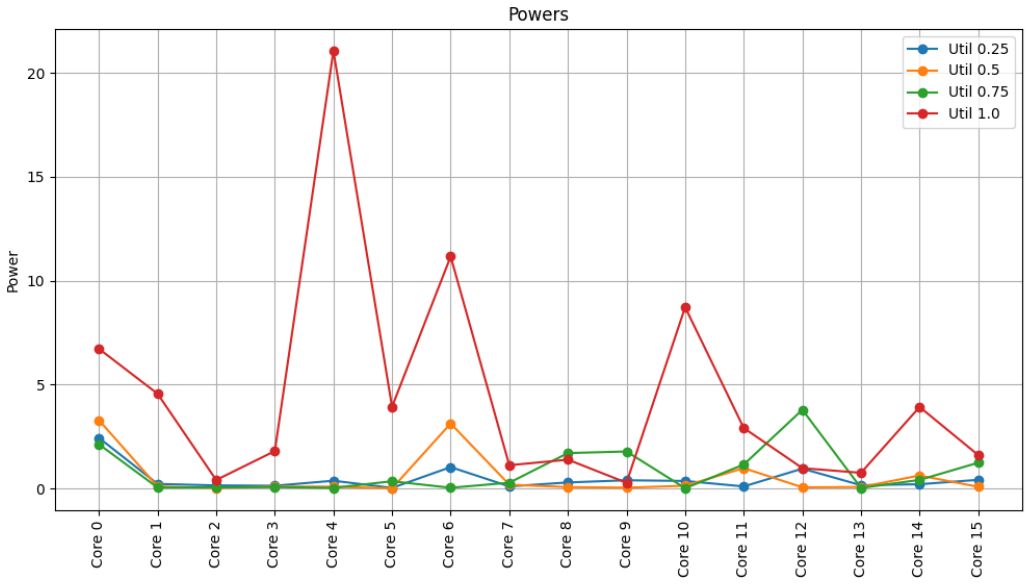
نمودار میزان کیفیت خدمات سامانه در حالت های مختلف:

```
def plot_system_quality_of_service(core, total_utilization, mean_qos_cores):
    plt.figure(figsize=(8, 5))
    plt.bar(total_utilization, mean_qos_cores, width=0.1, color='blue', alpha=0.7)
    plt.xlabel("Utilization")
    plt.ylabel("System QoS")
    plt.title("System QoS per Utilization")
    plt.xticks(total_utilization)
    plt.ylim(0, 1)
    plt.savefig(f"results/output2_core_{core}.png")
    plt.close()
```



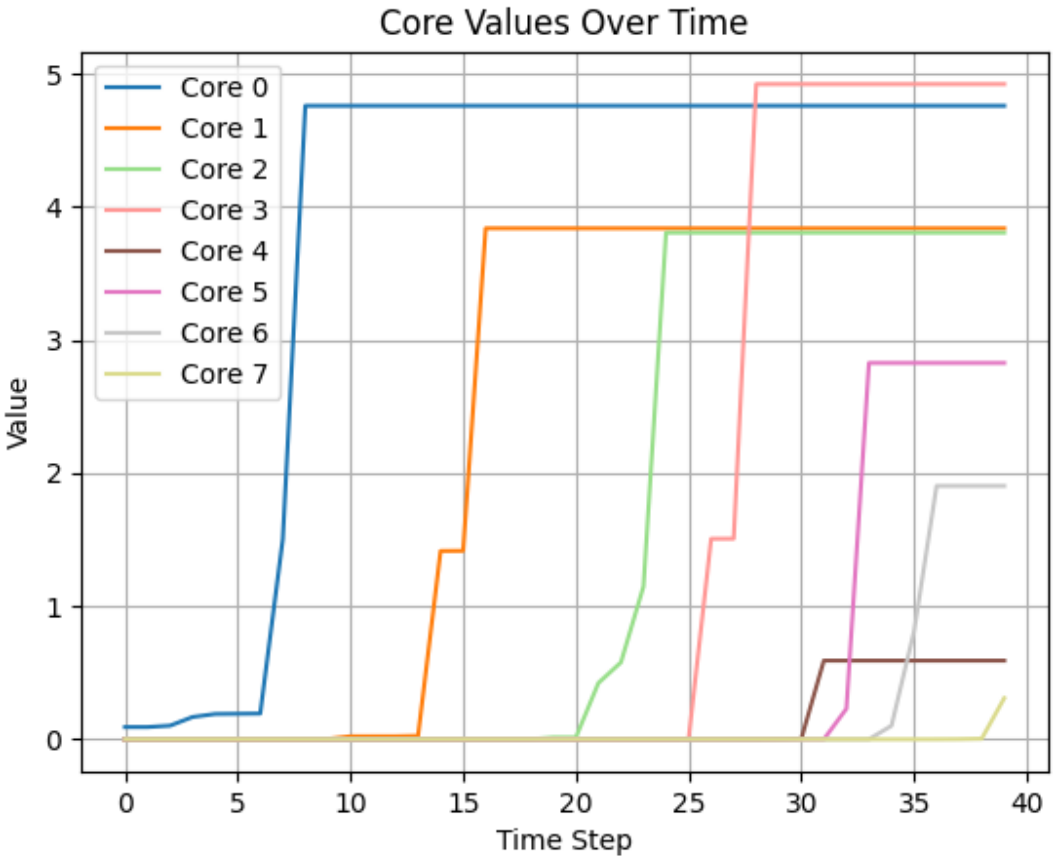
نمودار توان مصرفی هسته‌ها:

```python
def plot_power_cores(num_cores, total_utilization, powers_dict):
    x_labels = [f'Core {i}' for i in range(num_cores)]
    plt.figure(figsize=(12, 6))
    for utilization in total_utilization:
        plt.plot(x_labels, powers_dict[num_cores][utilization], marker='o', label=f'Util {utilization}')
    plt.title('Powers')
    plt.ylabel('Power')
    plt.xticks(rotation=90)
    plt.legend()
    plt.grid(True)

    plt.savefig(f"results/output3_with_{num_cores}_cores.png")
    plt.close()
```



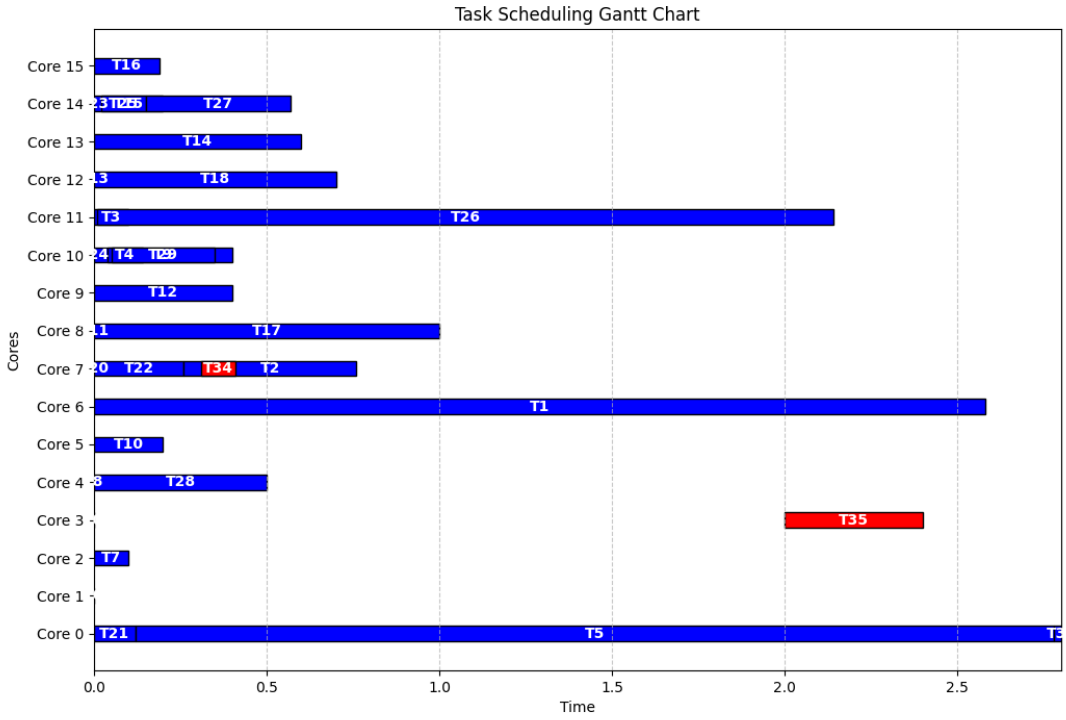
نمودار روند توان مصرفی در هر لحظه به ازای هر هسته:

```python
def plot_power_per_time(all_powers, num_cores, total_utilization):
    time_steps = list(range(len(all_powers)))
    core_values = {i: [entry[i] for entry in all_powers] for i in range(num_cores)}
    cmap = plt.get_cmap('tab20')
    colors = [cmap(i / num_cores) for i in range(num_cores)]
    for core in range(num_cores):
        plt.plot(time_steps, core_values[core], linestyle='-', color=colors[core], label=f'Core {core}')
    plt.xlabel('Time Step')
    plt.ylabel('Value')
    plt.title('Core Values Over Time')
    plt.legend()
    plt.grid(True)
    plt.savefig(f"results/output4_core_{num_cores}_utilization_{total_utilization}.png")
    plt.close()
```


Core Values Over Time
Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Value
Time Step
0
1
2
3
4
5
0
5
10
15
20
25
30
35
40

نمودار زمان بندی پذیری وظایف:

```python
def plot_gantt_chart(num_cores, total_utilization, core_tasks, max_hard_finish):
    fig, ax = plt.subplots(figsize=(12, 8))
    colors = {'hard': 'blue', 'soft': 'red'}  # رنگ‌های جداگانه برای hard و soft

    for core, tasks in core_tasks.items():
        for task in tasks:
            duration = task['finish'] - task['start']
            scaled_duration = scale_time(duration)

            ax.barh(y=core, width=scaled_duration, left=task['start'], height=0.4,
                    color=colors[task['type']], edgecolor='black')
            ax.text(task['start'] + scaled_duration / 2, core,
                    f'T{task["task_id"]}', ha='center', va='center', color='white', fontsize=10, fontweight='bold')

    ax.set_xlim(0, max_hard_finish)
    ax.set_xlabel('Time')
    ax.set_ylabel('Cores')
    ax.set_yticks(list(core_tasks.keys()))
    ax.set_yticklabels([f'Core {core}' for core in core_tasks.keys()])
    ax.set_title('Task Scheduling Gantt Chart')
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.savefig(f"results/output5_cores_{num_cores}_utilization_{total_utilization}.png")
    plt.close()
```



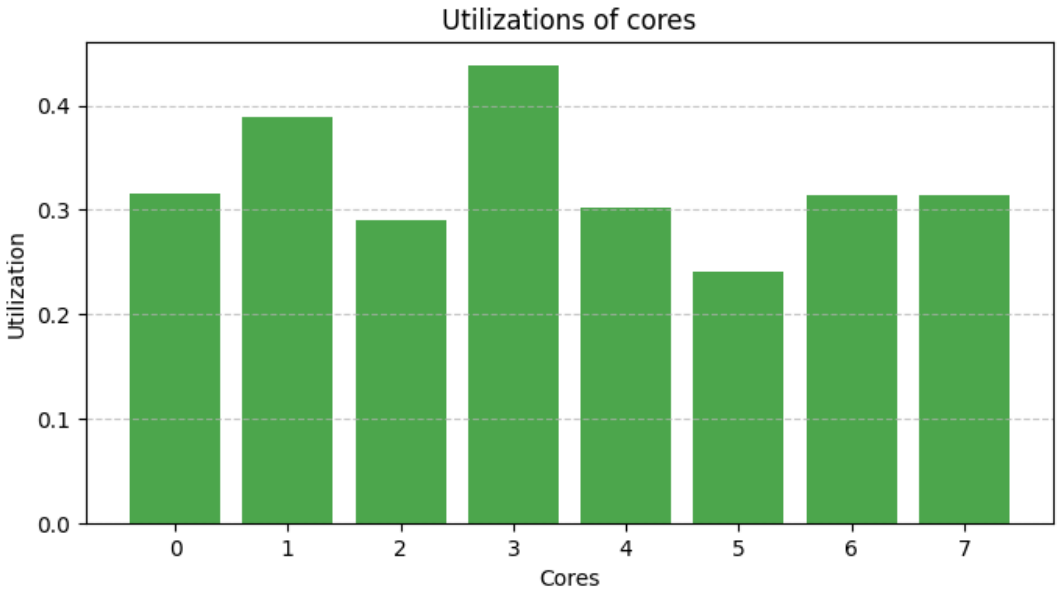
جدول وظایف و مشخصات آن‌ها:

برای این بخش خروجی **CSV** داریم که اطلاعات کامل تسک‌ها در آن وجود دارد.

```python
def output_tasks_table_csv(tasks):
    df = pd.DataFrame(tasks)
    df.set_index('task_id', inplace=True)
    df.index = [f'Task {i}' for i in df.index]
    df["wcet"] = round(df["wcet"], 4)
    df.to_csv(f'results/output6.csv')
```

	A	B	C	D	E	F	G	H	I	J
1		utilization	period	wcet	deadline	arrival_tim	type	frequency	voltage	
2	Task 0	0.1338	4	1.2845	4	0	hard	1.1	1.5	
3	Task 1	0.0285	5	3.4144	5	0	hard	0.75	0.9	
4	Task 2	0.0057	6	3.7932	6	0	hard	0.75	0.9	
5	Task 3	0.0147	1	0.6198	1	0	hard	0.75	0.9	
6	Task 4	0.0084	3	1.2931	3	0	hard	0.75	0.9	
7	Task 5	0.0011	1	0.0147	1	0	hard	0.75	0.9	
8	Task 6	0.0747	5	3.0887	5	0	hard	0.75	0.9	
9	Task 7	0.0238	2	0.0702	2	0	hard	0.75	0.9	
10	Task 8	0.041	3	0.2194	3	0	hard	0.75	0.9	
11	Task 9	0.0045	3	0.0239	3	0	hard	0.75	0.9	
12	Task 10	0.0401	3	0.7467	3	0	hard	0.75	0.9	
13	Task 11	0.0154	3	0.1211	3	0	hard	0.75	0.9	
14	Task 12	0.0018	6	0.0812	6	0	hard	0.75	0.9	
15	Task 13	0.0592	3	1.7761	3	0	hard	0.75	0.9	
16	Task 14	0.0083	2	0.1816	2	0	hard	0.75	0.9	
17	Task 15	0.0642	1	0.2705	1	0	hard	0.75	0.9	
18	Task 16	0.0681	5	3.4061	5	0	hard	0.75	0.9	
19	Task 17	0.0186	1	0.0225	1	0	hard	0.75	0.9	
20	Task 18	0.0227	5	1.0325	5	0	hard	0.75	0.9	
21	Task 19	0.0796	1	0.0965	1	0	hard	0.75	0.9	
22	Task 20	0.0045	6	0.2041	6	0	hard	0.75	0.9	

همچنین در تصویر زیر یک خروجی برای بهره‌وری را مشاهده می‌نمایید:



همچنین چند فایل text خروجی می‌دهیم که زمان‌بندی را به طور کامل‌تر نشان می‌دهد.

نمونه:

Core 0 (sorted by deadline):

Task 2 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 3

Task 10 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 3

Task 23 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 4

Task 8 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 6

Task 38 -> Type: soft, Arrival: 7, Start: 7.00, Finish: 7.09, Deadline: 8

Core 1 (sorted by deadline):

Task 9 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 2

Task 25 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 3

Task 0 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 4

Task 11 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 4

Task 21 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 4

Task 18 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 5

Task 31 -> Type: soft, Arrival: 1, Start: 1.00, Finish: 1.50, Deadline: 8

Core 2 (sorted by deadline):

Task 3 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 2

Task 12 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 2

Task 13 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 2

Task 19 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 3

Task 27 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 3

Task 28 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 6

Task 34 -> Type: soft, Arrival: 7, Start: 7.00, Finish: 7.08, Deadline: 10

Core 3 (sorted by deadline):

Task 1 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 4

Task 30 -> Type: soft, Arrival: 6, Start: 6.00, Finish: 7.19, Deadline: 10

Core 4 (sorted by deadline):

Task 29 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 2

Task 24 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 3

Task 4 -> Type: hard, Start: 0.00, Finish: 0.00, Deadline: 6

جمع‌بندی:

در این پروژه تلاش شد تا با کمک الگوریتم یادگیری تقویتی، زمان‌بندی آنلاین و آفلاین وظایف را انجام بدهیم. در فاز دوم، کاستی فاز اول که پیاده‌سازی TDP بود را کامل انجام داده و همچنین تمام خواسته‌های فاز دوم که همان فاز آنلاین می‌باشد را پیاده‌سازی نمودیم. در نهایت تمام خروجی‌ها و کد و گزارش را ارسال می‌نماییم. همچنین نکات کامل و تمامی خورجی‌ها با تست‌کیس‌های متفاوت، بعد از زمان ددلاین در صفحه گیتهاب Amirreza81 قرار خواهد گرفت.

پایان.