دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پروژه‌ی نهایی زمانبندی fpEDF

سیستم‌های بی‌درنگ

محمدرضا دولتی
۹۷۱۱۰۴۱۱

فرید فتوحی
۹۸۱۱۰۰۷۳

استاد:

دکتر سپیده صفری

تیر ۱۴۰۲

مقدمه:

در این پروژه قصد داریم که به کمک الگوریتم‌هایی که برای زمانبندی بر روی یک هسته مناسب می‌باشند، یک زمانبندی fpEDF انجام دهیم.

در این سامانه n وظیفه پریودیک داریم و پردازنده‌ای که m هسته پردازشی دارد، در ابتدا بدون در نظر گرفتن قسمت امتیازی پروژه فرض کردیم که تمامی پردازنده‌ها با فرکانس برابری مشغول به‌کار هستند و هر پردازنده یک بهره‌وری مشخص و برای تجمیع وظایف نیز یک بهره‌وری کل داریم.

الگوریتم fpEDF به‌طور کامل پیاده‌سازی می‌شود و تمامی خواسته‌های داک پروژه نیز برای شرح الگوریتم fpEDF نیز در نظر گرفته می‌شود.

همچنین برای رویکرد کلی پیاده‌سازی نیز مطابق با داک پروژه از الگوریتم UUnifast استفاده کردیم که بر اساس یک total utilization می‌آییم به‌صورت رندوم utilization هر تسک را به‌دست می‌آوریم که الگوریتم fpEDF را پیاده‌سازی و شبیه‌سازی نیز می‌کند.

در این پروژه نیز load balancing میان هسته‌ها نیز برقرار است و برای این کار نیز صفی را در نظر گرفته‌ایم که سیاست FCFS(Fisrt Come First Served) در آن در نظر گرفته شده است.

همچنین برای قسمت شی‌گرایی نیز دو شی در نظر گرفته‌ایم که یک شی برای Proccess ها و شی دیگر برای Task ها می‌باشد.

شیوه پیاده‌سازی:

کلاس Task: در این کلاس مقادیر لازم تعریف شده‌اند که آی‌دی، پریود، میزان بهره‌وری که قرار است تنها پارامتر ورودی ما باشد، ددلاین نسبی، زمان آزاد شدن و یا همان شروع هر تسک. زمان اتمام و پایان هر تسک، x که حاصل تقسیم ددلاین بر روی بهره‌وری(utilization) می‌باشد و c که حاصل ضرب پروید در بهره‌وری هر تسک اجرا شده می‌باشد.

کلاس Proccessor: در این کلاس نحوه‌ی تخصیص core ها به یکدیگر و تعداد آنها که در پروژه‌ی ما به‌صورت تصادفی از میان ۴ و ۸ و ۱۶ اتخاب می‌شود، نحوه‌ی تخصیص هسته‌ها و تعریف پارامترهای مورد نیاز در تصویر زیر آورده شده است:
umax در اینجا تعداد هسته‌های تخصیص داده شده که مثلا اگر single core باشیم، صفر است، بعلاوه‌ی ۱ تقسیم بر ۲ می‌باشد.

```
    def prepare(self, cc):
        """
        Creates empty partitioning table for Processor

        Parameters:

            cc: int -> core_count of Processor
        """
        self.core_dict = dict()

        for i in range(cc):
            self.core_dict[f"C{i+1}"] = dict()
            self.core_dict[f"C{i+1}"]["u"] = 0
            self.core_dict[f"C{i+1}"]["u_max"] = 1
            self.core_dict[f"C{i+1}"]["u_rel"] = 0
            self.core_dict[f"C{i+1}"]["Tasks"] = list()
```

در اینجا در واقع ما پارتیشن بندی‌ای برای هر Proccessor انجام دادیم که وضعیت جدول آن در ادامه برای اعمال زمانبندی بررسی می‌شود.
تابعی نیز تعریف کرده‌ایم برای اینکه تعداد هسته‌های لازم پردازنده را محاسبه کند و وظایف مورد نظر را به آنها تخصیص دهد:

```
    def get_necessary_cores_count(self):
        N = 0
        for i in range(self.core_count):
            if len(self.core_dict[f"C{i+1}"]["Tasks"]) != 0:
                N += 1
        return N
```

همچنین تابع global_EDF و الگوریتم fpEDF نیز در کلاس پردازنده تعریف شده‌اند، همانطور که می‌دانیم الگوریتم fpEDF که الگوریتمی است که تازه معرفی شده و نسبتا جدید نیز می‌باشد، به طور کلی از نوع گلوبال می‌باشد و در نحوه‌ی پیاده‌سازی کد ما این الگوریتم با توجه به پارامترهایی که در کلاس TaskSet تعریف شده‌اند، زمانبندی می‌شود و بعد از زمانبندی شدن که نحوه‌ی آن به طور کامل در TaskSet که بعدتر به آن می‌پردازیم، آورده شده است، بررسی می‌شود که آیا قابل زمانبندی شدن و یا خیر.

```python
def global_edf(self, T) -> bool:
    """
    Global Earliest Deadline First

    Parameters:

        T: TaskSet -> task set that should be scheduled

    Returns:

        bool -> True if scheduling was successful
    """
    self.reset()

    m = self.core_count
    T = T.sort('u')

    umax = T.taskset[-1].u

    u = umax + m * (1 - umax)

    print(f"u = {u}, Tu = {T.u}")

    print(f"Cores Necessary (N): {self.get_necessary_cores_count()}")

    return T.u < u
```

در قسمت آخر این کلاس و نحوه پیاده‌سازی آن نیز خود الگوریتم fpEDF و نحوه‌ی کارکرد آنرا تعریف کرده‌ایم که در اینجا هم وظایف با توجه مقدار بهره‌وریشان(utilization) مرتب سازی می‌شوند و تسک‌های با اولویت بالاتر تسک‌هایی هستند که u آنها بالاتر از 0.5 باشد و تسک‌های با اولویت پایینتر نیز تسک‌هایی‌اند که u کوچکتر مساوی 0.5 دارند.

که در آخر نیز تسک‌های سورت شده‌ی کوچکتر مساوی بین 0.5 و umax محاسبه می‌شوند.

```python
def fpedf(self, T) -> bool:
    """
    First Priority Earliest Deadline First

    Parameters:

        T: TaskSet -> task set that should be scheduled

    Returns:

        bool -> True if scheduling was successful
    """
    self.reset()

    m = self.core_count
    umax = (m + 1)/2

    T = T.sort('u')
    alpha = T.taskset[-1].u

    high_prio = [task for task in T.taskset if task.u > 0.5]
    low_prio = [task for task in T.taskset if task.u <= 0.5]

    print(f"High Priority Tasks: {high_prio}")
    print(f"Low Priority Tasks: {low_prio}")

    print(f"Cores Necessary (N): {self.get_necessary_cores_count()}")

    return alpha <= 0.5 and T.u <= umax
```

کلاس TaskSet: در این کلاس همه‌ی تعاریف مربوط به نحوه‌ی اجرای وظایف و تست کردن آنها آورده شده است. همچنین تابعی تعریف کرده‌ایم به نام NminElements که می‌آید لیست نهایی میان N لسیت یک وظیفه را برای ما مشخص می‌سازد.

```
def NminElements(list1, N):
    final_list = []

    for i in range(0, N):
        min1 = 0

        for j in range(len(list1)):
            if list1[j] < min1:
                min1 = list1[j]

        list1.remove(min1)
        final_list.append(min1)

    return final_list
```

در آخر نیز تابع main ای داریم که در ابتدا برای سهولت در انجام بهتر و زمانبندی و دید بهتری از پروژه مقادیر n و u ای برای آن تعریف کرده‌ایم که n تعداد تسک‌های مورد نظر و u مقدار مجموع بهره‌وری‌ای است که برای تمامی تسک‌ها در نظر داریم، همچنین نیز طبق داک پروژه که هر وظیفه پریودش ۱۰ ، ۲۰ یا ۱۰۰ در نظر گرفته می‌شود، به کمک تابع ranodm به‌وصورت کاملا رندوم پریودی را در نظر می‌گیریم:

```
##################################### main #################################
def main():
    n = int(input("please enter number of tasks: "))
    u = float(input("please enter total utlization: "))
    if u > n:
        print("Invalid input")
        return 0

    tasks = []
    sum_u = u
    next_sum_u = 0

    for i in range(1, n):
        next_sum_u = u * random.random() ** (1.0 / (n - i))
        tasks.append(task(utilization=sum_u - next_sum_u,period=random.choice(10,20,100),id=i))
        sum_u = next_sum_u

    tasks.append(task(utilization=sum_u,period=random.choice(10,20,100)),id=n)
```

برای اجرای نحوه‌ی درست زمانبندی نیز باید HyperPeriod ای تعریف کنیم که بدانیم وظایف را تا چه زمانی باید زمانبندی کنیم که برای این کار ب.ب.م پریود وظایف داده شده درون یک پردازنده را محاسبه می‌کنیم:

```
processors = []
x = 1
num_core = 16
for i in range(num_core):
    processors.append(Processor(id=x))
    x = x + 1
hyperperiod = tasks[0].period
for i in tasks[1:]:
    hyperperiod = int(hyperperiod * i.period / gcd(hyperperiod, i.period))

for i in tasks:
    i.set_c()

curent_time = 0
availabe_prosessors = []
run_prosessors = []
```

سپس برای هر Proccess آماده نیز ددلاین و x ای که قبلا نیز آنرا تعریف کرده‌ای ست می‌کنیم و تصویر کد پیاده‌سازی شده نیز آورده شده است:

```
for i in processors:
    if i.task == 0:
        availabe_prosessors.append(i.id)
for i in tasks:
    i.set_deadline(curent_time)
    i.set_x()
mux_c = NminElements(tasks,num_core)
for i in tasks:
    for j in mux_c:
        if i.id == j.id:
            i.run = True
            i.c = i.c - 1
            break
for i in range(num_core):
    availabe_prosessors[0].task = mux_c.id
    availabe_prosessors = availabe_prosessors.pop(0)

curent_time = curent_time + 1

while curent_time <= hyperperiod:
    for i in tasks:
        i.set_deadline(curent_time)
        i.set_x()
    mux_c = NminElements(tasks,num_core)
    for i in tasks:
        flag = 0
        for j in mux_c:
            if i.id == j.id:
                flag = 1
                i.run = True
                i.c = i.c - 1
                break
        if flag != 1:
            i.run = False
    curent_time = curent_time + 1
```

پیاده‌سازی نهایی کد و کلاس‌ها:

ابتدا تمامی کتابخانه‌هایی را که در روند پیاده‌سازی کد از آنها استفاده کردیم را import می‌کنیم:

```
import heapq
import selectors
import random
import matplotlib.pyplot as plt
from asyncio import tasks
from pickle import TRUE
from collections import deque
from math import gcd
from math import floor
```

در تصویر زیر نیز تعداد کل هسته‌ها و بهره‌وری کلی وظایف را نوشتیم: که نسبت سورت کردن آن به نسبت fpEDF متفاوت است زیرا در آنجا بر اساس ددلاین تقسیم بر میزان بهره‌وری بود ولی سورت انجام شده در این قسمت بر اساس فقط ددلاین می‌باشد.

```
N = list(range(1,17))
U = []
for i in range(1,17):
    U.append(random.uniform(0,i))
```

سپس تصاویر نحوه پیاده‌سازی و کد ها را در ادامه آوردیم که اول fpEDF را به کمک الگوریتم UUnifast که این الگوریتم در کلاس main نیز تعرریف شده است.

کلاس Task: در این کلاس مقادیر لازم تعریف شده‌اند که آی‌دی، پریود، میزان بهره‌وری که قرار است تنها پارامتر ورودی ما باشد، ددلاین نسبی، زمان آزاد شدن و یا همان شروع هر تسک. زمان اتمام و پایان هر تسک، x که حاصل تقسیم ددلاین بر روی بهره‌وری(utilization) می‌باشد و c که حاصل ضرب پروید در بهره‌وری هر تسک اجرا شده می‌باشد.

```
### fpEDF


sample_fpedf = []

class task:
    def __init__(self, period, utilization,id):
        self.id=id
        self.period = period
        self.utilization = utilization
        self.deadline = 0
        self.start=[]
        self.end=[]
        self.c =0
        self.x =0
        self.run = False

    def set_deadline(self, time):
        self.deadline = self.period -(time%self.period)
    def set_c(self,time):
        if time % self.period ==0 and self.c != 0:
            self.end.append(999)
            self.c=floor(self.utilization*self.period)
        elif time % self.period ==0 :
            self.c=floor(self.utilization*self.period)
    def set_x(self):
        self.x = self.deadline/self.utilization
```

کلاس Proccessor: در این کلاس نحوه‌ی تخصیص core ها به یکدیگر و تعداد آنها که در پروژه‌ی ما به‌صورت تصادفی از میان ۴ و ۸ و ۱۶ اتخاب می‌شود، نحوه‌ی تخصیص هسته‌ها و تعریف پارامترهای مورد نیاز در تصویر زیر آورده شده است:

```
class Processor:
    def __init__(self, id):
        self.id = id
        self.task = 0
        self.t = 0
        self.uti = 0
    def assign_task(self, task):
        self.task= task
    def set_uti(self,u):
        self.uti = self.uti + u
```

کد اصلی و تابع main که تعداد تسک‌ها و مقدار utilization ای که داده می‌شود و در چه صورتی این مقدار ممکن است invalid باشد و همچنین تعریف هایپرپریود که قرار است ب.م.م پریود وظایف را در نظر بگیرد و الگوریتم UUnifast و پیاده‌سازی آن نوشته شده است:

```
def main():
    for z in range(len(N)):
        n = 20
        u = U[z]
        if u > n:
            print("Invalid input")
            return 0

        tasks = []
        sum_u = u
        next_sum_u = 0

        for i in range(1, n):
            next_sum_u =  sum_u *random.random() ** (1.0 / (n - i))
            tasks.append(task(utilization=sum_u - next_sum_u,period=random.choice([10,20,100]),id=i))
            sum_u = next_sum_u

        tasks.append(task(utilization=sum_u,period=random.choice([10,20,100]),id=n))
        print(tasks[0])


        processors = []
        x = 1
        num_core = N[z]
        for i in range(num_core):
            processors.append(Processor(id=x))
            x = x + 1
        hyperperiod = tasks[0].period
        for i in tasks[1:]:
            hyperperiod = int(hyperperiod * i.period / gcd(hyperperiod, i.period))

        curent_time = 0


        for i in tasks:
            i.set_c(curent_time)


        while curent_time <= hyperperiod:
```

```
        for i in tasks:
            i.set_c(curent_time)

        for i in tasks:
            i.set_deadline(curent_time)
            i.set_x()

        tasks.sort(key = lambda y: (y.x))
        processors.sort(key=lambda y: (y.t),reverse=True)

        countr = 0
        temp = 0
        index = 0

        for i in range(n):
            temp = i
            if tasks[i].c > 0:
                tasks[i].run = True
                if tasks[i].c == floor(tasks[i].utilization*tasks[i].period):
                    tasks[i].start.append(curent_time)
                tasks[i].c = tasks[i].c - 1
                if tasks[i].c <=0:
                    tasks[i].end.append(curent_time)

                processors[index].task = tasks[i].id
                processors[index].t = 0
                processors[index].set_uti(1/tasks[i].period)
                index = index + 1

                countr = countr + 1
                if countr == num_core:
                    break
            else:
                tasks[i].run = False

           if temp != n - 1:
                temp = temp + 1
                for i in range(temp,n):
                    tasks[i].run = False


           for j in processors:
                if j.task == 0:
                    j.t = j.t + 1

           curent_time = curent_time + 1

       total_count = 0
       total_error = 0
       for h in tasks:
           total_count = total_count + (hyperperiod / h.period)
           if len(h.end) == 0:
               total_error = total_error + (hyperperiod / h.period)
               continue

           for k in h.end:
               if k == 999:
                   total_error = total_error + 1
       sample_fpedf.append(total_error/total_count)

   #     for i in processors:
   #         print(i.uti)
   #         print("_____")


if __name__ == "__main__":
    main()
```

خروجی‌ای که این کلاس به ما می‌دهد در واقع برای زمانی است که هر دفعه که شی تسک درست می‌کند، یک خروجی‌ای به ما داده می‌شود:

```
<__main__.task object at 0x7f87071f9e20>
<__main__.task object at 0x7f87071f9df0>
<__main__.task object at 0x7f87071f9e20>
<__main__.task object at 0x7f87071f9d60>
<__main__.task object at 0x7f87071f9df0>
<__main__.task object at 0x7f87071f9f10>
<__main__.task object at 0x7f87071f9f10>
<__main__.task object at 0x7f87071f9e50>
<__main__.task object at 0x7f87071f9d60>
<__main__.task object at 0x7f87071f9df0>
<__main__.task object at 0x7f87071f9f40>
<__main__.task object at 0x7f87071f9d60>
<__main__.task object at 0x7f87071f9f40>
<__main__.task object at 0x7f87071f9e20>
<__main__.task object at 0x7f87070af880>
<__main__.task object at 0x7f87070af880>
```

در قسمت fpEDF _global نیز کلاس‌های Task و Proccessor را داریم و تغییرات ایجاد شده مربوط به قسمت main کد این قسمت می‌باشد:

```
        for f in tasks:
            print(f.start,"",f.end)
        for h in tasks:
            total_count = total_count + (hyperperiod / h.period)
            if len(h.end) == 0:
                total_error = total_error + (hyperperiod / h.period)
                continue

            for k in h.end:
                if k == 999:
                    total_error = total_error + 1
        sample_global.append(total_error/total_count)

    #     for i in processors:
    #         print(i.uti)
    #         print("_____")


if __name__ == "__main__":
    main()
```

خروجی‌ای که برای این قسمت نیز داریم، زمان‌های شروع و پایان جاب خالی هر تسک می‌باشد که تصاویر این خروجی همگی در ادامه نیز آورده شده است:

```
[]  []
[]  []
[]  []
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[1, 21, 41, 61, 81]  [999, 1, 21, 41, 61, 81]
[2, 22, 42, 62, 82]  [999, 3, 23, 43, 63, 83]
[]  []
[]  []
[]  []
[]  []
[4]  [999, 13]
[14]  [999, 31]
<__main__.task object at 0x7f87071f5fd0>
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[0, 100]  [999, 0, 100]
[0, 100]  [999, 5]
[]  []
[]  []
[1]  [999, 1]
[2]  [999, 2]
[]  []
[3]  [999, 3]
<__main__.task object at 0x7f87071f5fd0>
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
[]  []
[]  []
[]  []
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[]  []
[]  []
[]  []
[1, 21, 41, 61, 81]  [999, 1, 21, 41, 61, 81]
[2]  [999, 5]
[2]  [999, 7]
[3]  [999, 3]
[4]  [999, 9]
[6]  [999, 13]
[8]  [999, 17]
[13]  [999, 59]
[]  []
<__main__.task object at 0x7f87071f5fd0>
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]  [999, 3, 13, 23, 33, 43, 53, 63, 73, 83, 93]
[]  []
[1, 21, 41, 61, 81]  [999, 3, 23, 43, 63, 83]
[1, 21, 41, 61, 81]  [999, 4, 24, 44, 64, 84]
[1, 21, 41, 61, 81]  [999, 6, 26, 46, 66, 86]
[]  []
[4, 24, 44, 64, 84]  [999, 4, 24, 44, 64, 84]
[4, 24, 44, 64, 84]  [999, 4, 24, 44, 64, 84]
[5]  [999, 29]
[5]  [999, 65]
[5]  [999, 37]
[7]  [999, 17]
[18]  [999, 56]
[34]  [999, 36]
[37]  [999, 45]
<__main__.task object at 0x7f87071f5fd0>
[]  []
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[]  []
[]  []
```

```
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[]  []
[1, 21, 41, 61, 81]  [999, 2, 22, 42, 62, 82]
[1]  [999, 2]
[1]  [999, 1]
[2]  [999, 8]
[2]  [999, 10]
[]  []
[3]  [999, 15]
[3]  [999, 7]
<__main__.task object at 0x7f87071f5fd0>
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[0, 100]  [999, 0, 100]
[]  []
[0, 100]  [999, 0, 100]
[0, 100]  [999, 0, 100]
[]  []
<__main__.task object at 0x7f87071f5fd0>
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[0, 20, 40, 60, 80, 100]  [999, 4, 24, 44, 64, 84]
[0, 100]  [999, 4]
[0, 100]  [999, 1]
[0, 100]  [999, 24]
[0, 100]  [999, 7]
[1]  [999, 11]
[2]  [999, 6]
[3]  [999, 12]
[5]  [999, 10]
<__main__.task object at 0x7f87071f5fd0>
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 99
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 6, 16, 26, 36, 46, 56, 66, 76, 86, 96]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[0, 20, 40, 60, 80, 100]  [999, 11, 31, 51, 71, 91]
[1, 21, 41, 61, 81]  [999, 6, 26, 46, 66, 86]
[]  []
[2]  [999, 39]
[]  []
[2]  [999, 10]
[3]  [999, 4]
[3]  [999, 7]
[5]  [999, 9]
[7]  [999, 72]
[7]  [999, 48]
<__main__.task object at 0x7f87071f5fd0>
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
```

```
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[]  []
[]  []
[0, 100]  [999, 8]
[]  []
[0, 100]  [999, 4]
[0, 100]  [999, 14]
[0, 100]  [999, 1]
<__main__.task object at 0x7f87071f5fd0>
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 4, 14, 24, 34, 44, 54, 64, 74, 84, 94]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 6, 26, 46, 66, 86]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[0, 100]  [999, 3]
[0, 100]  [999, 24]
[0, 100]  [999, 26]
[1]  [999, 7]
[1]  [999, 15]
[1]  [999, 17]
[2]  [999, 29]
[]  []
<__main__.task object at 0x7f87057c8160>
[]  []
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[]  []
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 3, 23, 43, 63, 83]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[0, 100]  [999, 21]
[0, 100]  [999, 24]
[0, 100]  [999, 2]
[0, 100]  [999, 3]
[0, 100]  [999, 16]
[]  []
[1]  [999, 20]
<__main__.task object at 0x7f87057c8160>
[]  []
[]  []
[]  []
[]  []
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[]  []
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[]  []
[]  []
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[0, 100]  [999, 3]
[]  []
[0, 100]  [999, 1]
[0, 100]  [999, 9]
[]  []
[]  []
[0, 100]  [999, 16]
<__main__.task object at 0x7f8707131c70>
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 8, 18, 28, 38, 48, 58, 68, 78, 88, 98]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
[0, 20, 40, 60, 80, 100]  [999, 3, 23, 43, 63, 83]
[0, 20, 40, 60, 80, 100]  [999, 999, 999, 999, 999, 999]
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[0, 20, 40, 60, 80, 100]  [999, 999, 999, 999, 999, 999]
[0, 20, 40, 60, 80, 100]  [999, 1, 21, 41, 61, 81]
[0, 20, 40, 60, 80, 100]  [999, 5, 25, 45, 65, 85]
[1, 21, 41, 61, 81]  [999, 8, 28, 48, 68, 88]
[1, 21, 41, 61, 81]  [999, 14, 34, 54, 74, 94]
```

```
[2]  [999, 70]
[2]  [999, 3]
[3]  [999, 999]
[3]  [999, 68]
[4]  [999, 45]
<__main__.task object at 0x7f8707036160>
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[]  []
[0, 100]  [999, 0, 100]
[]  []
[]  []
[0, 100]  [999, 0, 100]
[]  []
<__main__.task object at 0x7f87071f5fd0>
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 8, 18, 28, 38, 48, 58, 68, 78, 88, 98]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[0, 20, 40, 60, 80, 100]  [999, 16, 36, 56, 76, 96]
[0, 20, 40, 60, 80, 100]  [999, 7, 27, 47, 67, 87]
[0, 20, 40, 60, 80, 100]  [999, 999, 999, 999, 999, 999]
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[0, 20, 40, 60, 80, 100]  [999, 18, 38, 58, 78, 98]
[0, 20, 40, 60, 80, 100]  [999, 15, 35, 55, 75, 95]
[0, 100]  [999, 19]
[0, 100]  [999, 1]
[1]  [999, 999]
[1]  [999, 23]
[2]  [999, 23]
[1]  [999, 999]
[1]  [999, 23]
[2]  [999, 23]
[2]  [999, 4]
[2]  [999, 94]
<__main__.task object at 0x7f87071f5fd0>
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[]  []
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  [999, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[0, 20, 40, 60, 80, 100]  [999, 7, 27, 47, 67, 87]
[0, 20, 40, 60, 80, 100]  [999, 0, 20, 40, 60, 80, 100]
[]  []
[0, 20, 40, 60, 80, 100]  [999, 2, 22, 42, 62, 82]
[0, 20, 40, 60, 80, 100]  [999, 3, 23, 43, 63, 83]
[0, 100]  [999, 29]
[]  []
[]  []
[]  []
[0, 100]  [999, 25]
[0, 100]  [999, 2]
[0, 100]  [999, 7]
```

در آخر نیز نمودار نرخ زمانبند پذیری به تعداد هسته‌ها که یکی از آنها برای global fpEDF و دیگری برای fpEDF می‌باشد، رنگ نارنجی رنگ برای global fpEDF و رنگ آبی برای fpEDF می‌باشد: