به نام خدا

- **اعضای تیم:** ریحانه اخلاقیان، فروزان ایرجی، هانیه سادات میرعمادی

- **عنوان مقاله:** Systems Embedded Real-Time Heterogeneous on Technique Standby-Sparing Thermal-Aware

- **اهداف مقاله:** یک سیستم با m هسته که تعداد n مجموعه از وظایف بی‌درنگ نرم را اجرا می‌کند، داریم. مسئله‌ی ما این است که چگونه تخصیص وظایف به هسته‌ها، زمان‌بندی وظایف و سطح V-F هر وظیفه را پیدا کنیم تا به هدف اصلی دست یابیم. هدف اصلی روش ما این است که کیفیت خدمات سیستم را بیشینه کنیم؛ در حالی که مصرف برق هر هسته را تحت محدودیت TSP نگه داریم. این مسئله را می‌توان همانند فرمول شماره‌ی 8 در مقاله، فرمول‌بندی کرد.

- **محدودیت‌های موجود:**
  - ★ محدودیت توانی هسته: مصرف توان هر کدام از هسته‌ها در هر بازه‌ی زمانی t، باید کمتر از محدودیت TSP هسته باشد.
  - ★ محدودیت زمانی: زمان اتمام اجرای هر وظیفه باید زودتر از ددلاین آن وظیفه باشد.
  - ★ محدودیت تخصیص هسته: هر وظیفه فقط می‌تواند به یک هسته نگاشت شود.
  - ★ محدودیت قابلیت اطمینان وظیفه: از آنجا که ما از DVFS برای وظایف استفاده می‌کنیم، باید پس از DVFS، قابلیت اطمینان وظایف به حداقل مقدار ممکن برسد.
  - ★ محدودیت وابستگی: محدودیت‌های وابستگی بین وظایف نباید نقض شود؛ یعنی اگر بین دو وظیفه وابستگی وجود داشته باشد، زمان اتمام وظیفه‌ی قبلی باید قبل‌تر از زمان شروع وظیفه‌ی بعدی (وظیفه‌ی وابسته) باشد.

- **زبان برنامه‌نویسی:** پایتون

- **ابزارهای مورد نیاز:**
  1. gem5
  2. McPAT
  3. HotSpot
  4. TSP Protocol

- **خلاصه‌ی مقاله:** مصرف انرژی پایین و قابلیت اطمینان بالا از اهداف اصلی در طراحی سیستم‌های نهفته‌ی بی‌درنگ هستند. تکنیک standby-sparing می‌تواند قابلیت اطمینان سیستم را بهبود بخشد در حالی که ممکن است دمای سیستم را بیشتر از حد مجاز افزایش دهد. در این مقاله تکنیک TASS برای مدیریت توان مصرفی و قابلیت اطمینان در سیستم‌های چند هسته‌ای ناهمگن معرفی شده است. این روش با در نظر گرفتن محدودیت‌های زمانی و قابلیت‌اطمینان، وظایف را به گونه‌ای به هسته‌های مختلف تخصیص می‌دهد تا کارایی سیستم بهینه شود و از گرم شدن بیش از حد جلوگیری شود. این تکنیک علاوه بر اینکه خطاهای پایدار و گذرا را در سیستم‌های نهفته‌ی بی‌درنگ چند هسته‌ای تحمل می‌کند، محدودیت TSP را نیز به عنوان یک محدودیت انرژی در سطح هسته، برآورده می‌کند. با اجرای وظایف اصلی و پشتیبان روی هسته با هر مقدار مصرف انرژی‌ای که تحت محدودیت TSP باشد، سیستم بیش از حد گرم نمی‌شود. این روش از هم‌پوشانی اجرای تسک اصلی و پشتیبان برای مصرف کمتر انرژی جلوگیری می‌کند. برای بیشینه کردن کیفیت خدمات از پلتفرم‌های ناهمگن استفاده شده تا وظایف اصلی روی هسته‌هایی با کارایی بالا و توان زیاد اجرا شوند. وظایف پشتیبان روی هسته‌هایی با توان کم بعد از اتمام اجرای وظایف اصلی اجرا می‌شوند. اگر اجرای تسک اصلی به طور کامل با موفقیت انجام شود، نیازی به اجرای تسک پشتیبان نیست و همین باعث کاهش مصرف انرژی و دما می‌شود؛ در نتیجه در سناریوهای fault-free نیازی به استفاده از هسته‌ی یدک نیست و وظایف اصلی روی هسته‌ی اصلی زمان‌بندی و اجرا می‌شوند.

- **نحوه‌ی پیاده‌سازی گام به گام:**

1. تعریف مدل سیستم: ابتدا باید ساختار سیستم چند هسته‌ای ناهمگن را تعریف کرد که شامل هسته‌هایی با عملکرد بالا (HP) و هسته‌های کم مصرف (LP) است. این هسته‌ها به صورت جفتی برای اجرای وظایف استفاده می‌شوند.
2. نمایش وظایف: در سیستم، وظایف باید به صورت گراف‌های جهت‌دار بدون دور (DAG) نمایش داده شوند. هر گره نشان‌دهنده‌ی یک تسک با پارامترهای تعریف‌شده مانند زمان اجرای بدترین حالت بر روی هسته‌های HP و LP و ددلاین‌های مربوطه است.
3. محاسبه‌ی توان مصرفی: با استفاده از مدل‌های توان استاتیک و دینامیک، مصرف توان برای ترکیب‌های مختلف هسته‌ها برآورد می‌شود. معادلات کلیدی برای مصرف توان شامل پارامتر هایی مانند ولتاژ و فرکانس است که برای هر نوع هسته متفاوت است.
4. توان ایمن حرارتی (TSP): توان TSP به عنوان یک محدودیت توان بحرانی برای هر هسته تعیین می‌شود. TSP تضمین می‌کند که اجرای هسته زیر آستانه‌های نقض حرارتی باقی بماند و بر اساس تعداد هسته‌های فعال در زمان اجرا تنظیم شود.
5. زمان‌بندی و نگاشت وظایف: باید از سیاست زمان‌بندی LDF یا همان Last Deadline First برای تعیین اولویت‌های اجرای وظایف بر اساس ددلاین‌ها استفاده کرد. این کار برای برآورده کردن محدودیت‌های زمانی و بیشینه کردن کیفیت خدمات انجام می‌شود.
6. مدیریت در زمان اجرا: نظارت به اجرای وظایف باید به صورت مداوم صورت گیرد. وقتی که تسک اصلی با موفقیت کامل انجام شود، تسک پشتیبان مربوطه حذف می‌شود که باعث کاهش توان و حرارت

می‌شود. برای کاهش بیشتر مصرف توان می‌توان از DVFS برای استفاده از زمان اضافی موجود در اجرای وظایف استفاده کرد.

7. مدیریت خطا: باید یک مکانیزم مدیریت خطا مانند یک checker سخت‌افزاری را پیاده‌سازی کرد که به صورت مداوم قابلیت اطمینان وظایف را ارزیابی کند. در سناریوهایی که خطایی رخ نمی‌دهد، وظایف پشتیبان حذف می‌شوند و کارایی سیستم افزایش می‌یابد.
8. ارزیابی عملکرد: برای ارزیابی عملکرد نیز باید شبیه‌سازی کاملی از سیستم را با استفاده از بنچمارک‌های معتبر (مانند MiBench) انجام داد. می‌توان از ابزارهای gem5، McPAt و HotSpot برای ارزیابی معیارهای مختلفی از جمله کیفیت خدمات، مصرف توان، بیشترین توان و عملکرد حرارتی در سناریوهای واقعی و بدترین حالت استفاده کرد. در آخر نیز مقایسه‌ی TASS با روش‌های پیشرفته انجام می‌شود تا بهبودها با معیارهایی مانند کاهش بیشترین توان و کاهش دما به صورت دقیق اعتبارسنجی شود.

- **توضیح الگوریتم:** ابتدا باید وظایف مورد نیاز برای اجرا را تولید کنیم. طبق توضیحات گفته شده، ورودی وظایف به صورت یک گراف وابستگی (DAG) است که هر راس یک وظیفه با پنج پارامتر است:
  - موعد مقرر زمانی
  - بدترین زمان پردازش روی هسته با توان بیشتر
  - بدترین زمان پردازش روی هسته با توان کمتر
  - توان مصرفی تسک روی هسته با توان بیشتر
  - توان مصرفی تسک روی هسته با توان کمتر

از آنجایی که ما وظایف را تصادفی تولید می‌کنیم، دو پارامتر آخر تا زمانی که اطلاعاتی از هسته‌ها داشته باشیم، نمی‌توانند اضافه شوند. برای ساخت گراف وابستگی، از تابع *generate_dag* استفاده می‌کنیم. این تابع با گرفتن پارامترهای لازم برای تولید یک گراف تصادفی (چگالی، نظم و گسترش عرضی گراف) و تعداد رئوس (وظایف)، گراف مورد نظر را تولید می‌کند. با دستورات زیر می‌توانیم گراف وابستگی را برای ۱۰ وظیفه در متغیر dag داشته باشیم:

```
n_tasks = 10
density = 0.4
regularity = 0.6
fatness = 0.4

dag = generate_dag(n_tasks, density, regularity, fatness)
```

الگوریتم برای ساخت گراف و تخصیص ددلاین‌ها به وظایف از طریق گراف‌هایی با ویژگی‌های مختلف طراحی شده است. همان‌طور که پیش‌تر گفته شد، در این الگوریتم، ابتدا گراف با استفاده از پارامترهایی مانند تعداد وظایف (tasks)، چگالی (density)، نظم (regularity)، فتنس (fatness) و برخی ویژگی‌های دیگر ساخته می‌شود. ساخت گراف به‌طور کلی به فرآیند تعریف روابط بین وظایف و تخصیص شاخه‌ها به آن‌ها مربوط می‌شود.

یکی از ویژگی‌های مهم این الگوریتم، جلوگیری از هم‌پوشانی زمانی وظایف اصلی و backup آنها است. به عبارت دیگر، زمان‌بندی به‌گونه‌ای انجام می‌شود که وظایف به‌طور هم‌زمان روی هسته‌ها اجرا نشوند و زمان‌بندی‌ها کاملاً مستقل از یک‌دیگر باشد. همچنین، ددلاین نسبی وظایف باید از مجموع ددلاین‌های دو هسته بیشتر باشد، به‌گونه‌ای که وظایف به‌طور هماهنگ و با توجه به توان مصرفی اختصاص‌یافته به هسته‌ها، زمان‌بندی شوند.

برای تخصیص ددلاین‌ها، ابتدا از یک مقدار اسلک فکتور (slack factor) استفاده می‌شود که به‌صورت تصادفی بین 1.5 تا 3 انتخاب می‌شود. این اسلک فکتور، مقداری اضافی به زمان تخصیص داده می‌شود تا اطمینان حاصل شود که وظایف در زمان مناسب و بدون تداخل با وظایف دیگر به پایان برسند.

در این الگوریتم، ددلاین‌ها ابتدا به‌طور مستقل برای هر وظیفه محاسبه می‌شوند، سپس این ددلاین‌ها با توجه به ددلاین‌های وظایف والد (parent tasks) تنظیم می‌شوند. ددلاین هر وظیفه نباید از ددلاین وظیفه‌ی والد کمتر باشد. در صورتی که این شرایط رعایت نشود، ددلاین وظیفه‌ی به‌روز رسانی شده و به ددلاین وظیفه‌ی والد نزدیک‌تر می‌شود.

در نهایت، الگوریتم با استفاده از اسلک فکتور جدید، ددلاین‌ها را به‌روزرسانی کرده و از روش‌هایی مانند جستجوی درخت والدین (parent nodes) برای جلوگیری از مشکلات هم‌پوشانی ددلاین‌ها و اطمینان از رعایت ترتیب زمانی استفاده می‌کند. این فرایند به‌طور مداوم ادامه می‌یابد تا زمانی که تمام وظایف به‌طور صحیح و مطابق با ددلاین‌های تعیین‌شده زمان‌بندی شوند.

***دقت شود که:

1. تخصیص ددلاین‌ها باید از مجموع ددلاین‌های دو هسته بیشتر باشد.

2. زمان‌بندی وظایف باید از هم‌پوشانی زمانی جلوگیری کند.

3. ددلاین‌ها باید در هر مرحله به‌روز شوند تا از هم‌خوانی با ددلاین‌های وظایف والد اطمینان حاصل شود.

4. اسلک فکتور به‌صورت تصادفی برای تنظیم زمان تخصیص استفاده می‌شود.

این الگوریتم در نهایت باعث می‌شود که تمامی وظایف به‌طور مؤثر و بهینه بر روی هسته‌ها توزیع شوند، بدون اینکه از ددلاین‌های تعیین‌شده عقب بیفتند.

- کلاس ***Corepair***: این کلاس برای مدیریت هسته‌های پردازشی و زمان‌بندی وظایف است:

- *__init__*: مقداردهی اولیه هسته‌های پرقدرت و کم‌قدرت، تعیین شناسه یکتا و وضعیت فعال/غیرفعال بودن هسته‌ها.
- *get_utilization*: محاسبه‌ی بهره‌وری سیستم (پیاده‌سازی نشده).
- *find_first_free_time_slot_after*: پیدا کردن اولین بازه‌ی زمانی آزاد بعد از زمان k (فعلاً فقط k+1 را بر می‌گرداند).
- *get_tsp*: محاسبه‌ی پارامتر زمان‌بندی وظایف (در حال حاضر مقدار ثابت 100 را بر می‌گرداند).
- *schedule*: زمان‌بندی یک وظیفه و ذخیره‌ی آن در لیست زمان‌بندی، همراه با چاپ پیام تأیید.

- کلاس ***System***: این کلاس برای مدیریت جفت هسته‌های پردازشی و کنترل وضعیت فعال/غیرفعال هر هسته است.
  - *__init__:*
    - *self.islands*: لیستی برای ذخیره‌ی جفت هسته‌های پردازشی.
    - *self.total_time*: تعیین زمان کلی عملکرد سیستم (200 واحد زمانی).
  - *add_core_pair:* اضافه کردن یک جفت هسته‌ی پردازشی به لیست islands.
  - *activate:* پیدا کردن جفت هسته با pair_id مشخص.
    - core=0: فعال‌سازی هسته‌ی پرقدرت (is_high_active=True).
    - core=1: فعال‌سازی هسته‌ی کم‌قدرت (is_low_active=True).
  - *deactivate:* مشابه متد *activate*، اما برای غیرفعال‌سازی هسته‌ها:
    - core=0: غیرفعال‌سازی هسته‌ی پرقدرت.
    - core=1: غیرفعال‌سازی هسته‌ی کم‌قدرت.

- کلاس ***TaskScheduler***: این قسمت، وظایف را با توجه به محدودیت‌های انرژی و وابستگی‌های وظایف روی جفت‌های هسته پردازشی زمان‌بندی می‌کند و از الگوریتم‌های صف اولویت و کنترل مصرف انرژی بهره می‌برد.
  - *__init__:*
    - *core_pairs*: لیستی از جفت‌های هسته پردازشی.
    - *G*: گراف وظایف که وابستگی بین وظایف را نشان می‌دهد.
    - *leaves*: لیست وظایف بدون وابستگی.
    - *priority_queue*: صف اولویت‌بندی وظایف برای زمان‌بندی.
  - *return_leaves:* پیدا کردن وظایفی که هیچ وابستگی‌ای ندارند و اضافه کردن آنها به leaves.

- *make_priority_queue:* برگ‌های گراف را پیدا می‌کند. از بین آن‌ها، وظیفه‌ای با مهلت دیرتر (latest deadline) را انتخاب می‌کند. آن وظیفه را به صف اولویت (priority_queue) اضافه کرده و از گراف حذف می‌کند. این روند را تا خالی شدن گراف ادامه می‌دهد.
- *min_utilization:* انتخاب جفت هسته‌ای که کمترین میزان استفاده را دارد (فعلاً فقط اولین جفت هسته را برمی‌گرداند).
- *schedule_task:* زمان‌بندی وظایف طبق مراحل زیر:
  - پیدا کردن برگ‌های گراف.
  - انتخاب وظیفه‌ای با مهلت دیرتر.
  - زمان‌بندی روی هسته‌ی اصلی اگر محدودیت انرژی (TSP) اجازه دهد.
  - در صورت نیاز، زمان‌بندی وظیفه روی هسته‌ی یدکی برای افزایش قابلیت اطمینان (Bi-scheduling).

توابع کمکی:

- *get_cores:* دریافت اطلاعات هسته‌ها از شبیه‌ساز gem5 و تبدیل آنها به فرمت مورد نیاز.
- *assign_tasks_power_consumption(G, core_pairs):* تخصیص میزان مصرف انرژی به وظایف در گراف، براساس ویژگی‌های هسته‌های پردازشی.
- *get_TSP:* دریافت محدودیت‌های انرژی TSP از ابزار Hotspot.

- تابع ***draw_dag***: این تابع، گراف ساخته‌شده (5 پارامتر گفته شده در ابتدای توضیحات) را نمایش می‌دهد. این تابع برای رسم یک گراف جهت‌دار بدون حلقه با استفاده از کتابخانه‌های NetworkX و Matplotlib طراحی شده است. این تابع، گراف ورودی را به شکلی خوانا نمایش می‌دهد.

*مراحل اصلی عملکرد تابع:

1. ایجاد بوم رسم: تنظیم اندازه‌ی شکل برای نمایش بهتر گراف.
2. محاسبه‌ی موقعیت گره‌ها: استفاده از الگوریتم Kamada-Kawai برای چیدمان بهینه‌ی گره‌ها.
3. رسم گره‌ها و یال‌ها: گره‌ها به رنگ آبی روشن و با اندازه مشخص رسم می‌شوند. یال‌ها به رنگ خاکستری و دارای فلش جهت‌دار هستند.
4. اضافه کردن برچسب‌ها: نمایش اطلاعات هر گره شامل برچسب، مقدارهای کمینه و بیشینه‌ی زمان اجرا (WC_low, WC_high) و موعد مقرر زمانی (deadline).
5. تنظیمات نهایی: حذف محورها، تنظیم عنوان گراف، و نمایش نهایی.

در نتیجه‌ی مراحل بالا، گرافی واضح، مرتب و قابل درک از ساختار گره‌ها و یال‌های یک DAG ساخته می‌شود.

- توضیحات تابع ***print_dag_stats***: این قسمت برای تولید، تحلیل و زمان‌بندی وظایف در یک گراف جهت‌دار بدون دور طراحی شده است.

مراحل کلی اجرای این کد به شرح زیر است:

1. محاسبه‌ی ویژگی‌های **DAG**:

   این تابع ویژگی‌های مختلفی از گراف را محاسبه می‌کند:

   تعداد گره‌ها :(**num_nodes**) تعداد کل وظایف.

   تعداد یال‌ها :(**num_edges**) تعداد وابستگی‌های بین وظایف.

   میانگین درجه‌ی گره‌ها :(**avg_degree**) نسبت تعداد یال‌ها به تعداد گره‌ها.

   طول مسیر بحرانی :(**critical_path**) طولانی‌ترین مسیر وابستگی در DAG.

   تعداد سطوح (**levels**): سطوح مختلف بر اساس مرتب‌سازی توپولوژیکی.

2. تولید **DAG:**

   تولید یک DAG با پارامترهای:

number_of_tasks: تعداد وظایف (با توجه به مقاله ۷ مرتبه شبیه‌سازی را برای تعداد وظایف ۳۶، ۴۵، ۵۴، ۶۳، ۷۲، ۸۱ و ۹۰ انجام می‌دهیم و در نهایت نتایج را با هم مقایسه می‌کنیم)

0.4 = ***density***: میزان چگالی ارتباط بین وظایف (تعداد یال‌های بین رئوس به نسبت بیشترین تعداد ممکن یال‌ها)

0.6 = ***regularity***: میزان نظم ساختار گراف (نظم بیشتر یعنی رئوس به طور متوازن پخش شده‌اند و گراف بالانس بیشتری دارد)

0.4 = ***fatness***: میزان پراکندگی گره‌ها در سطوح مختلف (در هر سطح چه تعداد راس وجود دارد و در واقع چه تعداد وظیفه موازی با هم اجرا می‌شوند)

3. نمایش گراف و ویژگی‌های وظایف تولید شده و وابستگی‌شان:

drow_dag(dag, “Generated DAG”)

- کتابخانه‌ی **NX**:

کتابخانه‌ی **NX** برای ایجاد گراف‌های بدون دور و جهت‌دار بسیار مفید است و به راحتی می‌تواند گراف‌هایی با ویژگی‌های دلخواه را تولید کند. سپس، زمان‌بندی وظایف بر اساس تخصیص ددلاین‌ها صورت می‌گیرد. در این بخش، زمان‌بندی به‌گونه‌ای است که دو هسته در نظر گرفته می‌شود: یک هسته با توان مصرفی پایین و دیگری با توان مصرفی بالاتر. زمان‌بندی وظایف به گونه‌ای انجام می‌شود که وظایف با توان مصرفی بالا، ابتدا بر روی هسته با توان بالاتر و سپس بر روی هسته با توان مصرفی پایین‌تر قرار می‌گیرند.

قابل ذکر است که کدهای پروژه نیز در آدرس زیر وجود دارد:

https://github.com/Fonij80/embedded-system-university-project

- **نحوه‌ی ارزیابی الگوریتم:**

۱. **گام اول: نصب برنامه‌های مورد نیاز برای ارزیابی الگوریتم**

- **Gem5**: ابتدا کتابخانه‌های مورد نیاز این برنامه را نصب می‌کنیم (بسته به ورژن اوبونتو کتابخانه‌های مورد نیاز متفاوت است):

- Ubuntu 24.04: gem5 >= v24.0

  *$ sudo apt install build-essential scons python3-dev git pre-commit zlib1g zlib1g-dev *
  *libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev *
  *libboost-all-dev libhdf5-serial-dev python3-pydot python3-venv python3-tk mypy *
  m4 libcapstone-dev libpng-dev libelf-dev pkg-config wget cmake doxygen
- Ubuntu 22.04: gem5 >= v21.1

  *$ sudo apt install build-essential git m4 scons zlib1g zlib1g-dev *
  *libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev *
  python3-dev libboost-all-dev pkg-config python3-tk
- Ubuntu 20.04: gem5 >= v21.0

  *$ sudo apt install build-essential git m4 scons zlib1g zlib1g-dev *
  *libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev *
  *python3-dev python-is-python3 libboost-all-dev pkg-config gcc-10 g++-10 *
  python3-tk

  سپس با توجه به ورژن اوبونتو، نسخه‌ی gem5 سازگار با آن را از آدرس زیر دانلود می‌کنیم:

  https://github.com/gem5/gem5/releases

  برای بیلد کردن gem5 با معماری ARM ابتدا کتابخانه‌های زیر را نصب می‌کنیم:

  $ sudo apt-get install gcc-arm-linux-gnueabihf gcc-aarch64-linux-gnu device-tree-compiler
  $ sudo apt install scons

$ pip install -r requirements.txt

برای بیلد کردن پروژه با معماری ARM دستور زیر را در فولدر پروژه اجرا می‌کنیم:

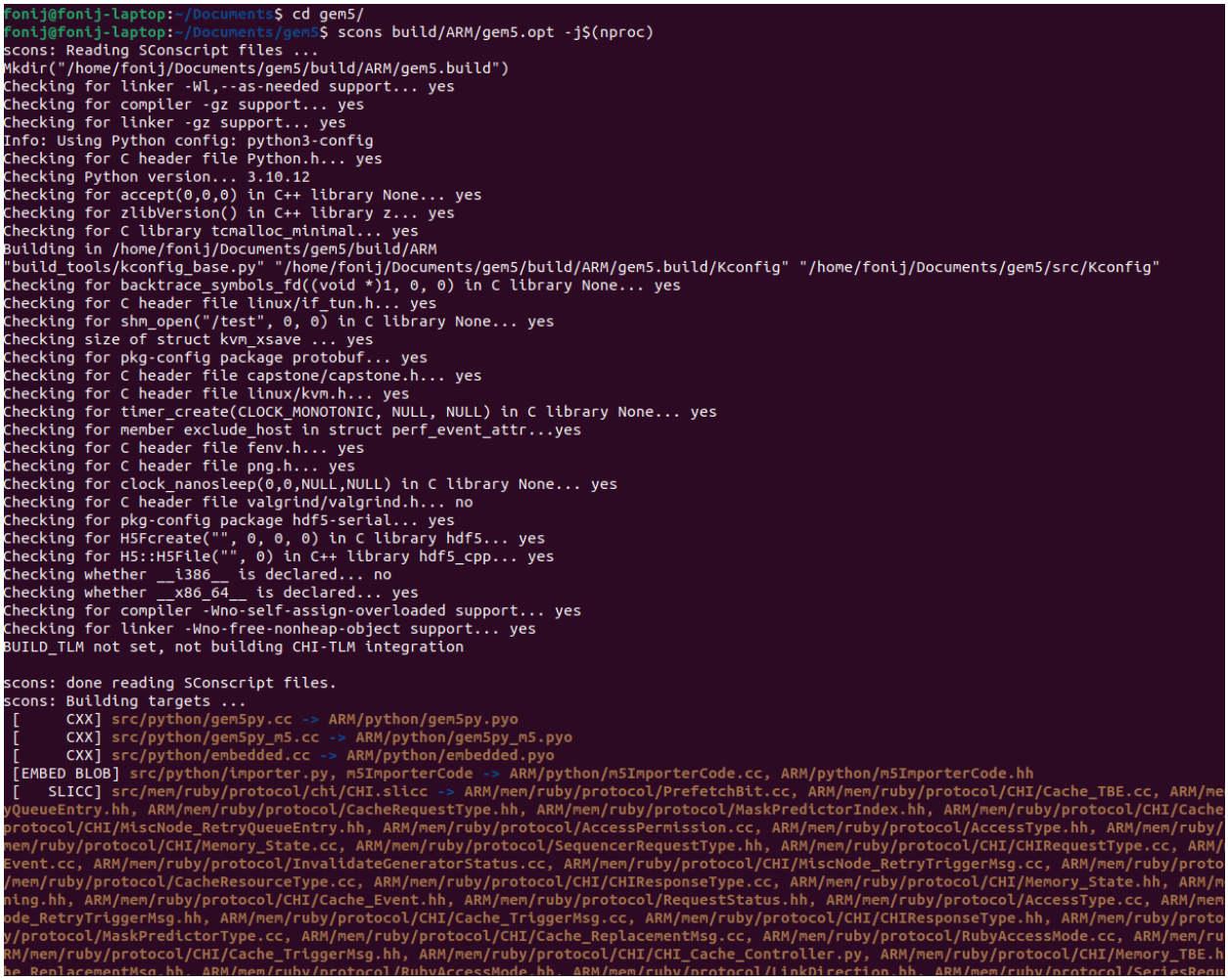
$ scons build/ARM/gem5.opt -j$(nproc)

فرایند بیلد اولیه‌ی gem5 طولانی است و ممکن است روی برخی از سیستم‌ها تا دو ساعت هم طول بکشد. همچنین یکی از خطاهای محتمل در فرایند بیلد، خطای ناسازگاری ورژن protocol buffer با gem5 است که برای gem5 v24.1.0.1 باید حتما ورژن 3.21 یا بالاتر پروتکل بافر روی سیستم نصب باشد. برای نصب پروتکل بافر هم ابتدا باید برنامه‌ی زیر را با زدن دستورات زیر در ترمینال، نصب کنیم:

$ git clone https://github.com/abseil/abseil-cpp.git
$ cd abseil-cpp
$ mkdir build && cd build
$ cmake ..
$ make -j$(nproc)
$ sudo make install
$ cmake -DABSL_ROOT_DIR=/path/to/abseil-cpp ..

$ git clone https://github.com/protocolbuffers/protobuf.git
$ cd protobuf
$ mkdir build && cd build
$ cmake ..
$ make -j$(nproc)
$ sudo make install
$ sudo ldconfig

نتیجه‌ی بیلد کردن gem5:



برای نحوه‌ی کار دقیق با gem5 می‌توانید به مستند زیر مراجعه کنید:

https://www.gem5.org/documentation/

- **McPAT**: ابتدا پروژه‌ی McPAT را از آدرس زیر کلون می‌کنیم:

https://github.com/HewlettPackard/mcpat.git

کتابخانه‌های مورد نیاز زیر را برای کامپایل کردن برنامه نصب می‌کنیم:

$ sudo apt install build-essential g++-multilib libc6-dev-i386 libc6-dev

سپس برای کامپایل کردن برنامه، دستور *make* را در داخل فولدر پروژه‌ای که کلون کردیم، اجرا می‌کنیم.

در صورتی که قبل از آن کامپایل ناموفقی انجام داده‌ایم، باید ابتدا دستور *make clean* را اجرا کنیم تا تمام فایل‌های اشتباه قبلی پاک شوند تا موقع کامپایل دوباره به کانفلیکت نخوریم.

در نهایت با اجرای دستور زیر، اگر کامپایل‌مان با موفقیت انجام شده باشد، نحوه‌ی استفاده از برنامه در ترمینال نشان داده می‌شود.

$ ./mcpat -h

خروجی‌ای که باید ببینیم:

How to use McPAT:
mcpat -infile <input file name> -print_level < level of details 0~5 > -opt_for_clk < 0 (optimize for ED^2P only)/1 (optimzed for target clock rate)>

لازم به ذکر است که این برنامه از ابتدا برای محیط‌های Unix-based ساخته شده و برای بهره بردن از تمام قابلیت‌های آن بهتر است که روی سیستم عامل‌های Unix-based نصب شود.

```
(base) fonij@fonij-laptop:~/Documents/mcpat$ ./mcpat -h
How to use McPAT:
  mcpat -infile <input file name>  -print_level < level of details 0~5 >  -opt_f
or_clk < 0 (optimize for ED^2P only)/1 (optimzed for target clock rate)>
```

- **HotSpot:** ابتدا پروژه را از آدرس زیر کلون می‌کنیم:

  https://github.com/uvahotspot/HotSpot.git

  کتابخانه‌های مورد نیاز زیر را نصب می‌کنیم:

  $ sudo apt install libblas-dev libsuperlu5 libsuperlu-dev

  سپس دستور زیر را برای نصب این برنامه اجرا می‌کنیم:

  $ cd HotSpot && make SUPERLU=1

  برای اجرای شبیه‌سازی نیز دستورات زیر را اجرا می‌کنیم:

  $ chmod +x run.sh
  $ ./run.sh

  نمونه خروجی شبیه‌سازی:

```
(base) fonij@fonij-laptop:~/Documents/HotSpot/examples$ cd example1
(base) fonij@fonij-laptop:~/Documents/HotSpot/examples/example1$ chmod +x run.sh
(base) fonij@fonij-laptop:~/Documents/HotSpot/examples/example1$ ./run.sh
Parsing input files...
Creating thermal circuit...
Computing temperatures for t = 0.000000e+00...
Computing temperatures for t = 1.000000e-02...
Computing temperatures for t = 2.000000e-02...
Computing temperatures for t = 3.000000e-02...
Computing temperatures for t = 4.000000e-02...
Computing temperatures for t = 5.000000e-02...
Computing temperatures for t = 6.000000e-02...
Computing temperatures for t = 7.000000e-02...
Computing temperatures for t = 8.000000e-02...
Computing temperatures for t = 9.000000e-02...
Computing temperatures for t = 1.000000e-01...
Computing temperatures for t = 1.100000e-01...
Computing temperatures for t = 1.200000e-01...
Computing temperatures for t = 1.300000e-01...
Computing temperatures for t = 1.400000e-01...
Computing temperatures for t = 1.500000e-01...
Computing temperatures for t = 1.600000e-01...
Computing temperatures for t = 1.700000e-01...
Computing temperatures for t = 1.800000e-01...
Computing temperatures for t = 1.900000e-01...
Computing temperatures for t = 2.000000e-01...
Computing temperatures for t = 2.100000e-01...
Computing temperatures for t = 2.200000e-01...
Computing temperatures for t = 2.300000e-01...
Computing temperatures for t = 2.400000e-01...
Computing temperatures for t = 2.500000e-01...
Computing temperatures for t = 2.600000e-01...
Computing temperatures for t = 2.700000e-01...
Computing temperatures for t = 2.800000e-01...
Computing temperatures for t = 2.900000e-01...
Computing temperatures for t = 3.000000e-01...
Computing temperatures for t = 3.100000e-01...
Computing temperatures for t = 3.200000e-01...
Computing temperatures for t = 3.300000e-01...
Computing temperatures for t = 3.400000e-01...
Computing temperatures for t = 3.500000e-01...
Computing temperatures for t = 3.600000e-01...
Computing temperatures for t = 3.700000e-01...
Computing temperatures for t = 3.800000e-01...
Computing temperatures for t = 3.900000e-01...
Computing temperatures for t = 4.000000e-01...
Computing temperatures for t = 4.100000e-01...
Computing temperatures for t = 4.200000e-01...
Computing temperatures for t = 4.300000e-01...
Computing temperatures for t = 4.400000e-01...
```

۲. گام دوم: ارزیابی الگوریتم با برنامه‌های نصب‌شده

برای ارزیابی الگوریتم TASS یک سیستم چند هسته‌ای ناهمگن که شامل دو نوع هسته با کارایی بالا (HP) و با کارایی پایین (LP) است، داریم. به تعداد k = m/2 جفت هسته که هر کدام شامل یک هسته‌ی اصلی و یک هسته‌ی اضافی (برای اجرای وظیفه‌ی backup) است. وظایف اصلی (Ti) به هسته‌ی اصلی و کپی آن وظیفه به هسته‌ی اضافی مپ می‌شوند. توان مصرفی این سیستم برابر جمع توان پویا و توان ایستا است و با فرمول زیر محاسبه می‌شود.

$$P_{total}(V_i, f_i) = P_{static} + P_{dynamic} = I_0 e^{\frac{-V_{th}}{\eta V_T}} V_i + \alpha_i C_L V_i^2 f_i$$

همچنین گراف جهت‌دار بدون دوری (DAG) برای وظایف بی‌درنگ نرم با سطح بحرانیت یکسان داریم که وابستگی وظیفه‌ها را به هم نشان می‌دهد و همچنین هر وظیفه بعد از اجرای کامل وظیفه‌ی پدرش اجرا می‌شود. هر وظیفه با سه پارامتر $\{WC_i^{LP}, WC_i^{HO}, D_i\}$ مشخص می‌شود که به ترتیب نشان‌دهنده‌ی موعد زمانی وظیفه، بدترین زمان اجرا روی هسته با کارایی بالا (وظیفه‌ی اصلی) و بدترین زمان اجرا روی هسته با کارایی پایین (وظیفه‌ی پشتیبان) است.

ترتیب مراحل ارزیابی الگوریتم و نحوه‌ی کار با نرم‌افزارها

1. بعد از بیلد کردن موفقیت‌آمیز gem5، باید اسکریپت configuration ای برای اجرای شبیه‌سازی و به دست آوردن اطلاعات هسته‌ها و کارایی‌شان (داده‌های آماری مربوط به معماری سیستم) با این نرم‌افزار بنویسیم. ابتدا ماژول m5 را به محیط پایتون‌مان با دستورات زیر اضافه می‌کنیم (آدرس فولدر پایتون پروژه‌ی gem5 ای که کلون کردیم را در دستور زیر قرار می‌دهیم):

$ export PYTHONPATH=/path/to/gem5/src/python:$PYTHONPATH
$ source ~/.bashrc

سپس اسکریپت configuration ای که برای gem5 نوشته‌ایم را با دستور زیر اجرا می‌کنیم. (کد آن در فایل TASS_Config.py در کدهای پروژه‌ی پیوست‌شده وجود دارد.)

$ build/ARM/gem5.opt configs/TASS_Config.py

بعد از اجرای شبیه‌سازی gem5 باید خروجی‌های زیر را در فولدر **m5out** داشته باشیم:

- **stats.txt:** این فایل شامل جزئیات آماری شبیه‌سازی از جمله معیارهای کارایی، فعالیت هسته‌ها و دسترسی‌های حافظه است.
- **config.json:** این فایل به فرمت جیسون شامل اطلاعاتی مانند معماری، کانفیگ‌های هسته، سایز کش و دیگر کانفیگ‌های مورد نیاز سیستم است.

تعریف سیستم و هسته‌ها در *TASS_Config.py*:

```
system = System()
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()

# Create heterogeneous CPU clusters
system.cluster0 = [ArmMinorCPU(cpu_id=i + 2) for i in range(2)]  # HP Cortex-A15 cores
system.cluster1 = [ArmO3CPU(cpu_id=i) for i in range(2)]  # LP Cortex-A7 cores
```

تعریف حافظه و اتصال هسته‌ها به bus:

```
# Set up memory system
system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]
system.membus = SystemXBar()

# Connect CPUs to memory bus
for cpu in system.cluster0 + system.cluster1:
    cpu.createInterruptController()
    cpu.connectAllPorts(system.membus)
```

تخصیص پردازه به هسته‌ها و تخصیص الگوریتم زمان‌بندی به سیستم:

```
# Create workload
process = Process()
process.cmd = ['path/to/benchmark']
for cpu in system.cluster0 + system.cluster1:
    cpu.workload = process

system.tass_scheduler = TASS_Implementation.TaskScheduler(system)
```

2. برای به دست آوردن توان‌های مصرفی (ایستا و پویا) هر هسته، ابتدا باید خروجی‌های gem5 یعنی فایل‌های stats.txt و config.json را با اسکریپت پایتونی پارس کنیم و اطلاعات مورد نیاز McPAT را به فرمت xml ای که این نرم‌افزار به عنوان ورودی می‌گیرد، به دست آوریم. این اطلاعات شامل تعداد هسته‌ها، سایز کش، الگوی دسترسی به حافظه و تعداد دستورات می‌باشد. بعد از تولید فایل xml طبق فرمت ورودی McPAT، دستور زیر را برای اجرای این نرم‌افزار در ترمینال می‌زنیم:

$ ./mcpat -infile input.xml -print_level 2

برای پارس کردن خروجی‌های gem5 می‌توان از پروژه‌ی زیر استفاده کرد. دستورات زیر را برای گرفتن ورودی‌های McPAT در ترمینال اجرا می‌کنیم:

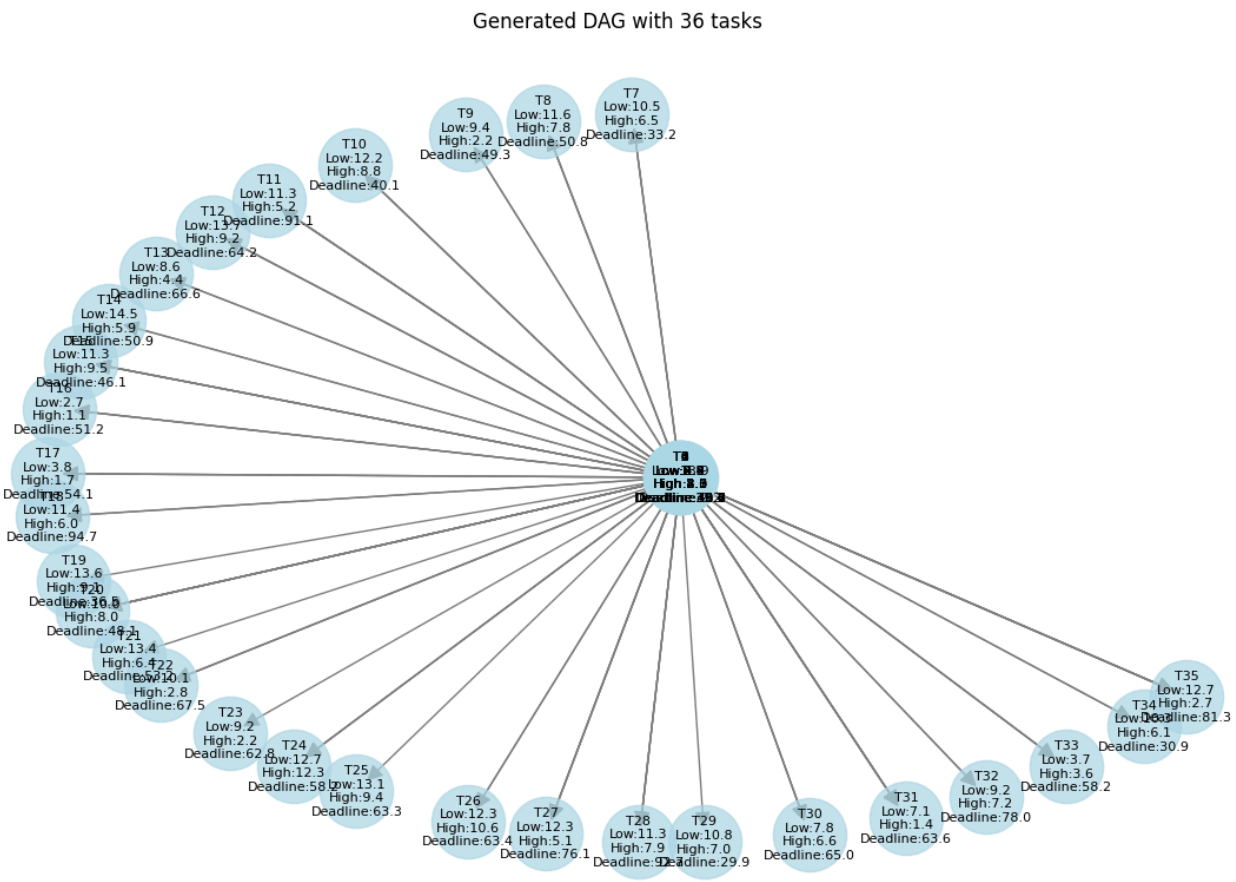
$ git clone https://github.com/Hardik44/Gem5toMcPat_parser
$ make
$ python Program.py stats.txt config.json> mcpat-template.xml

در نهایت McPAT میزان توان ایستا و پویای مصرفی هر هسته را در خروجی‌اش مشخص می‌کند. در این خروجی باید اوج توان مصرفی هر هسته را مشخص کنیم و ببینیم که آیا همچنان توان‌ها در محدوده‌ی TSP قرار می‌گیرند یا نه.

3. از خروجی McPAT به عنوان ورودی HotSpot استفاده می‌کنیم و تغییرات دمایی هسته‌ها را به دست می‌آوریم و مشاهده می‌کنیم که TASS بهتر از روش‌های پیشین دمای هسته‌ها را کنترل می‌کند.
در نهایت برای ارزیابی دقیق‌تر روش TASS بهتر است آن را با گراف وظیفه‌های مختلفی تست کنیم، همچنین می‌توان از برنامه‌های مجموعه‌ی MiBench هم که روی سیستم‌های چندهسته‌ای ناهمگن اجرا می‌شوند نیز استفاده کرد. (استفاده از هسته‌ی ARM Cortex-A7 به عنوان هسته‌ی LP و استفاده از هسته‌ی ARM Cortex-A15 به عنوان هسته‌ی HP)

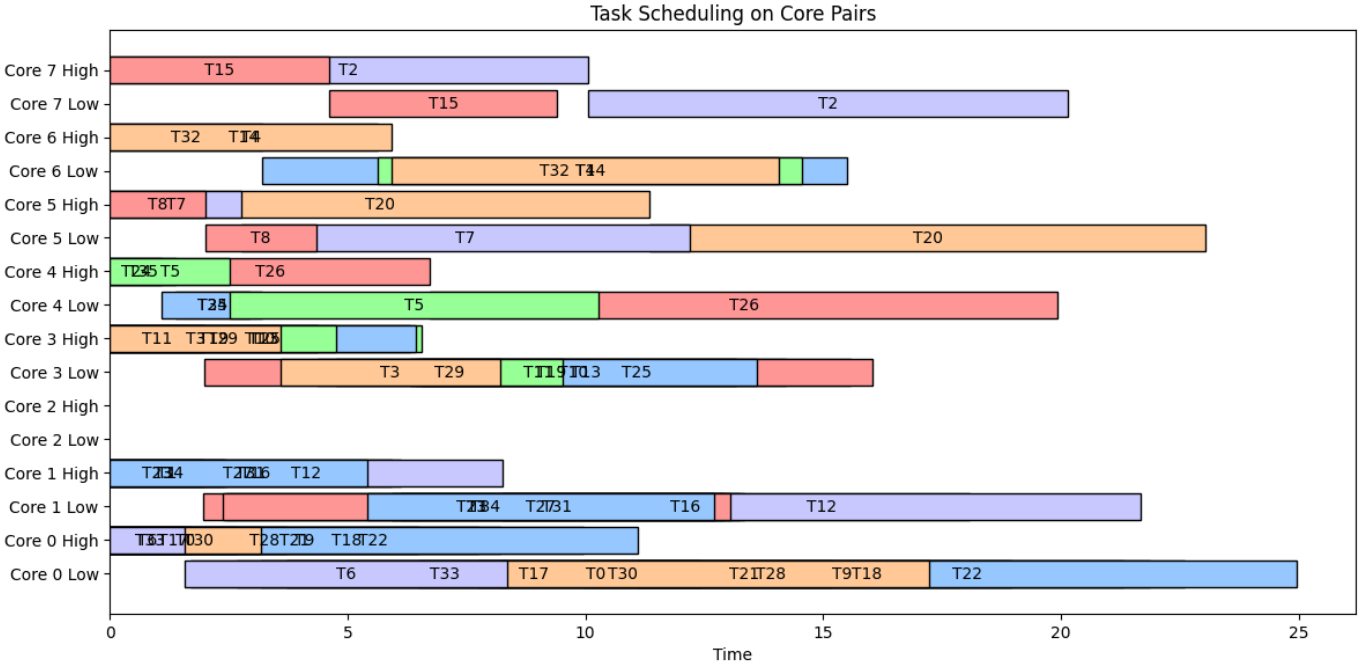
- **بررسی نتایج:** مانند شکل شماره‌ی ۵ مقاله، زمان‌بندی را برای ۷ مجموعه وظیفه با تعداد وظایف ۳۶، ۴۵، ۵۴،۶۳، ۷۲، ۸۱ و ۹۰ انجام می‌دهیم.

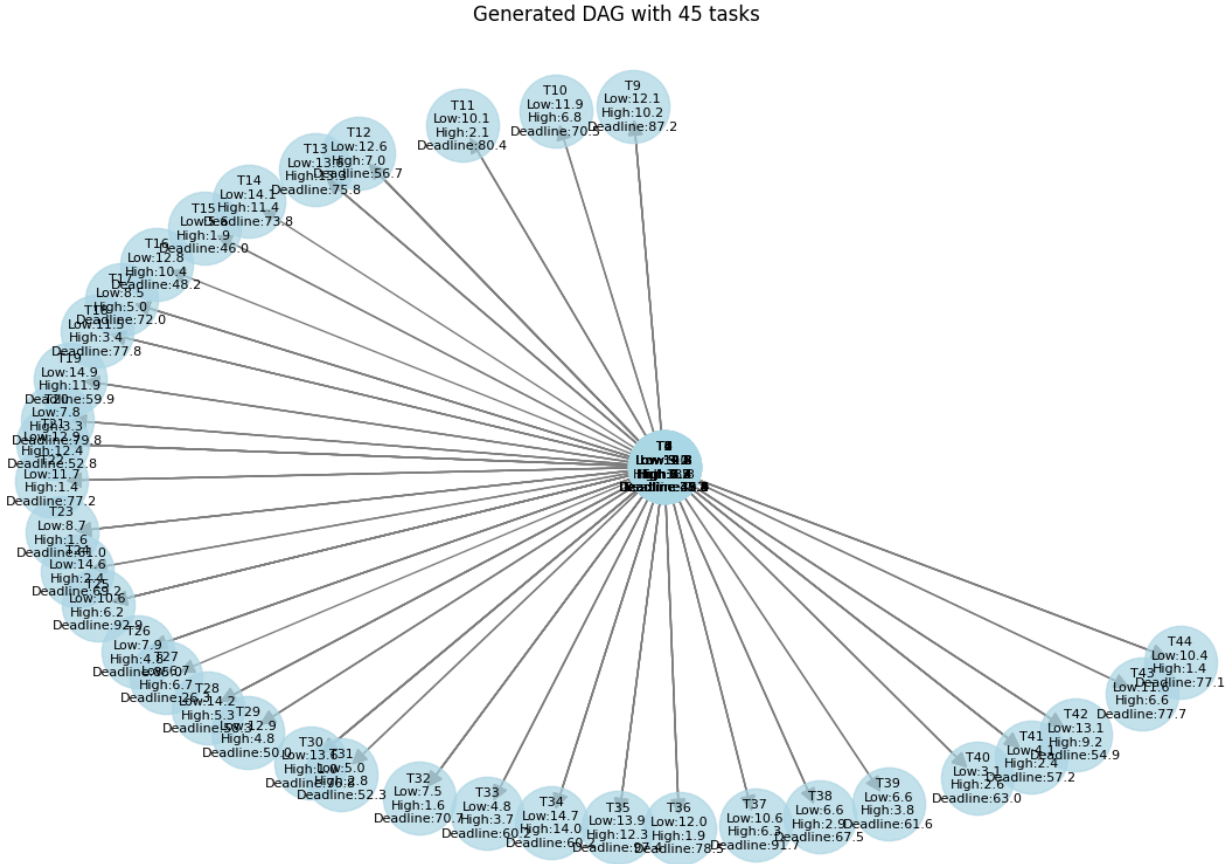
گراف وظایف (DAG) برای هر مجموعه وظیفه به صورت زیر است. بر روی هر راس ددلاین، بدترین زمان اجرا روی هسته‌ی HP و بدترین زمان اجرا روی هسته‌ی LP نوشته شده است. میزان توان مصرفی برای هر جفت هسته نیز به صورت عدد رندمی بین حداقل و حداکثر توان مصرفی آن هسته اختصاص یافته است. در تابع ***get_cores*** برای تولید جفت هسته‌ها برای هسته‌ی LP (ARM Cortex-A7) عددی بین ۱۰۰ تا ۳۵۲ میلی‌وات (حداقل و حداکثر توان مصرفی این هسته) و برای هسته‌ی HP (ARM Cortex-A15) عدد رندمی بین ۵۰۰ تا ۱۰۰۰ میلی‌وات به هسته‌ی تولیدشده به عنوان توان مصرفی اختصاص می‌یابد.



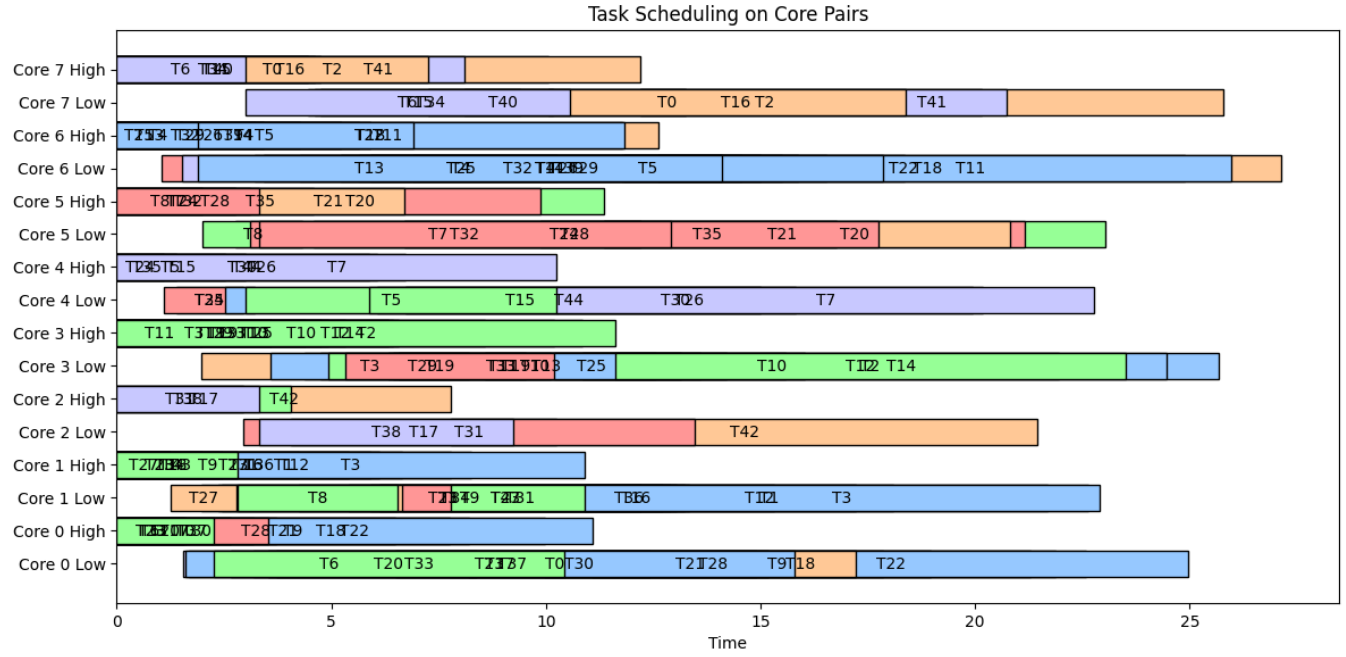
DAG Statistics: num_nodes: 36, num_edges: 81, avg_degree: 2.25, critical_path: 1, levels: 2

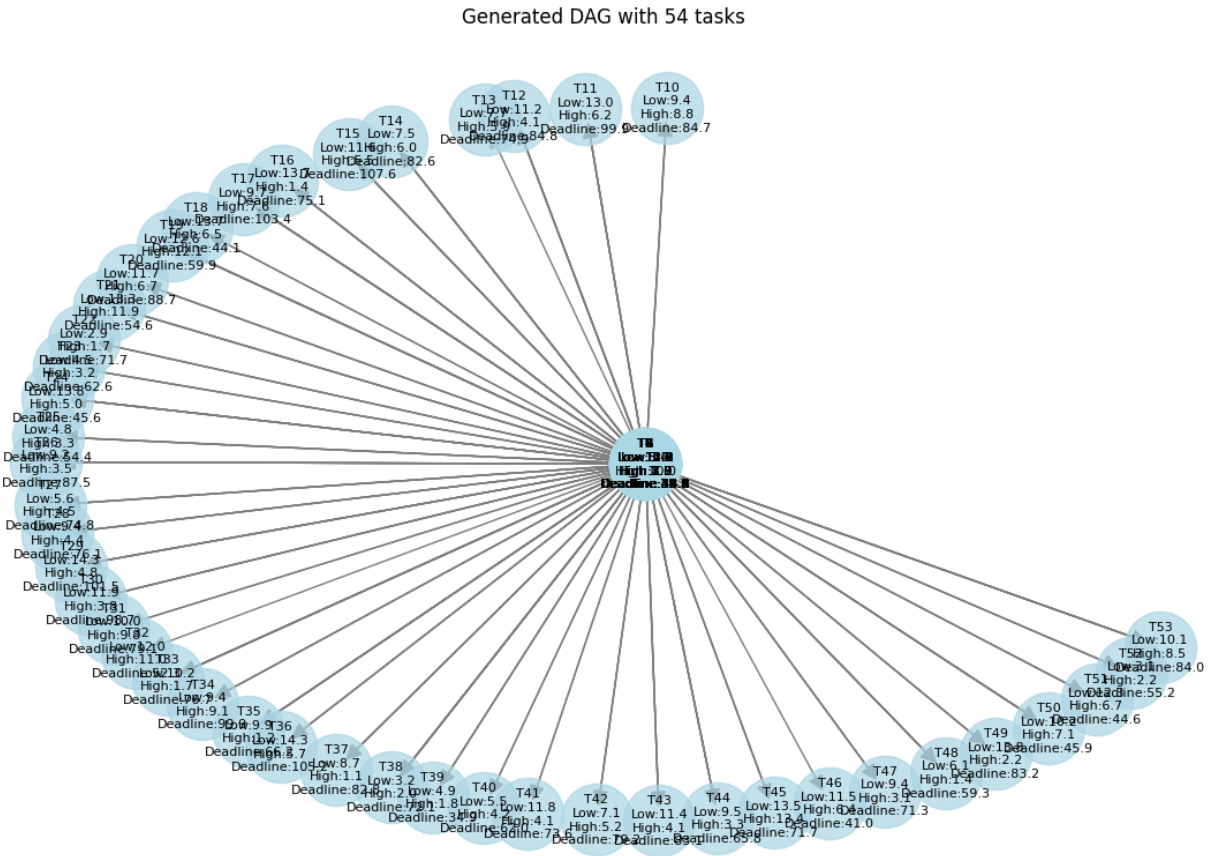
نمودار زمان‌بندی نیز بر روی جفت هسته‌های اصلی و پشتیبان برای ۳۶ تا وظیفه به صورت زیر است (روی هر نمودار نام وظیفه‌ی در حال اجرا نوشته شده است) در زمان‌بندی از الگوریتم LDF استفاده شده است و تا جای ممکن سعی شده که هم‌پوشانی بین وظایف در حال اجرا روی هسته‌های اصلی و پشتیبان کم شود تا مصرف توان نیز کمتر شود.



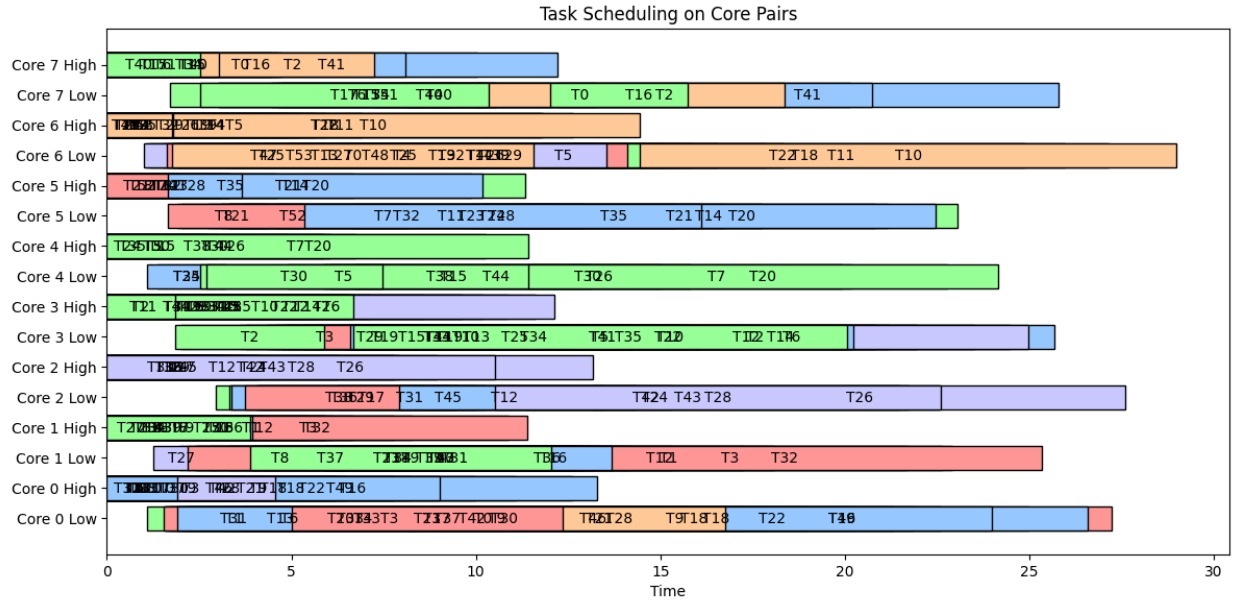


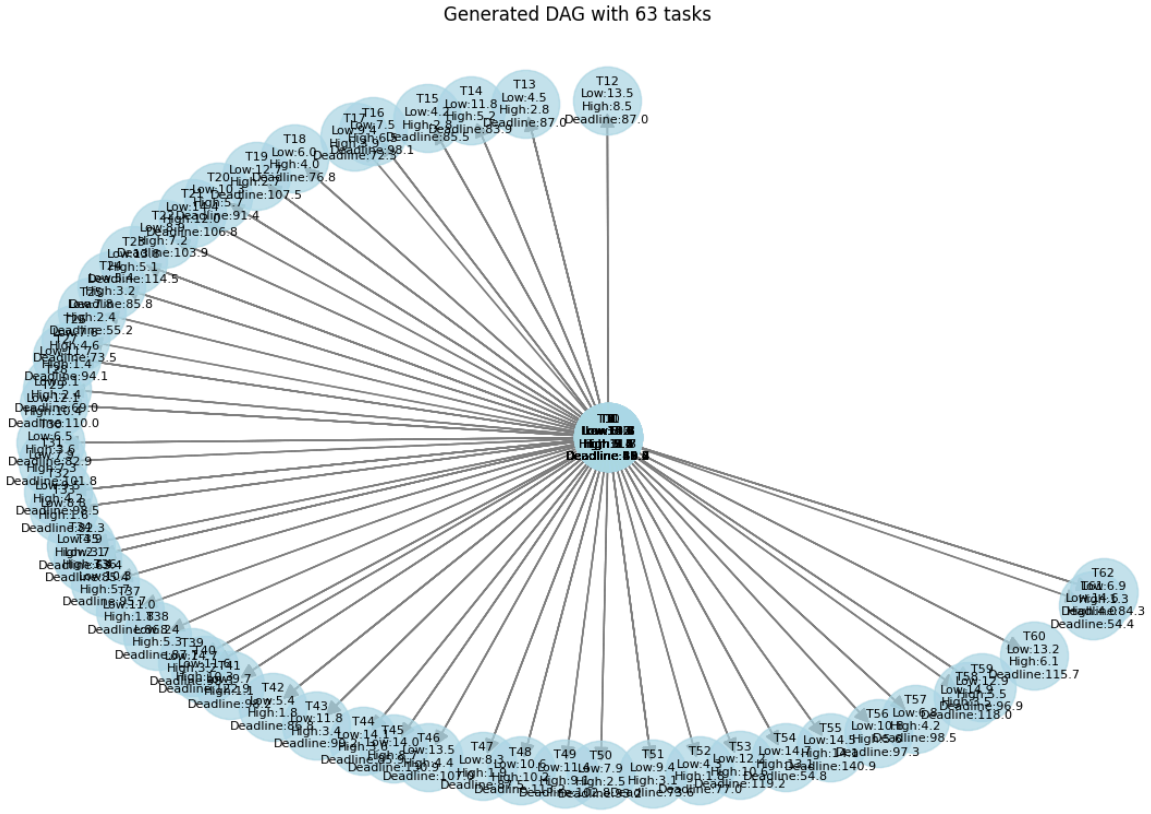
DAG Statistics: num_nodes: 45, num_edges: 130, avg_degree: 2.888888888888889, critical_path: 1, levels: 2



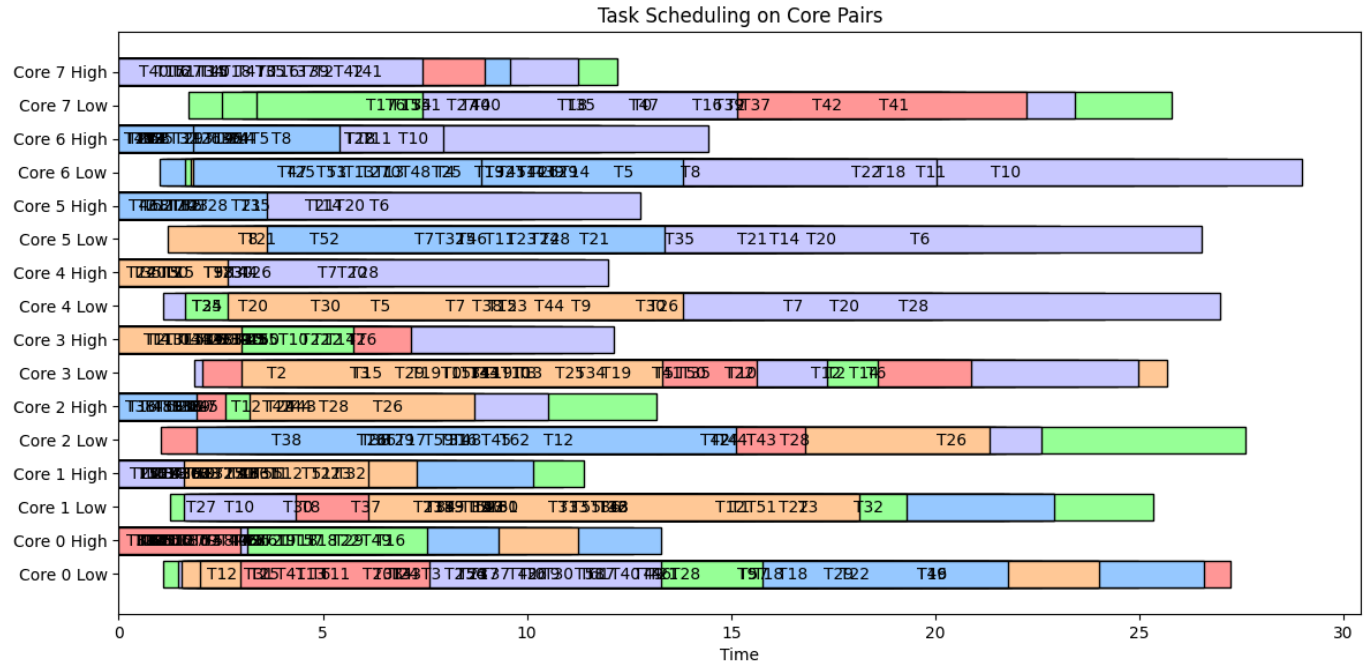


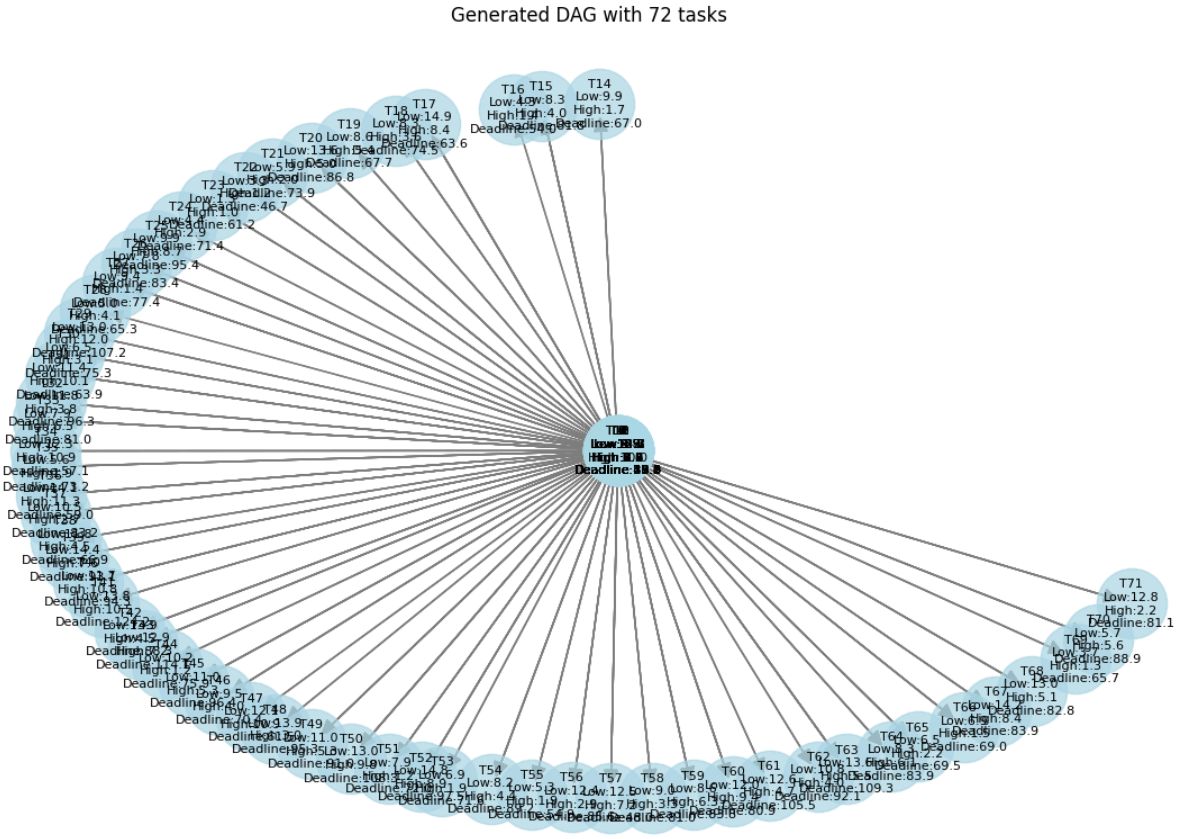
DAG Statistics: num_nodes: 54, num_edges: 175, avg_degree: 3.240740740740741, critical_path: 1, levels: 2



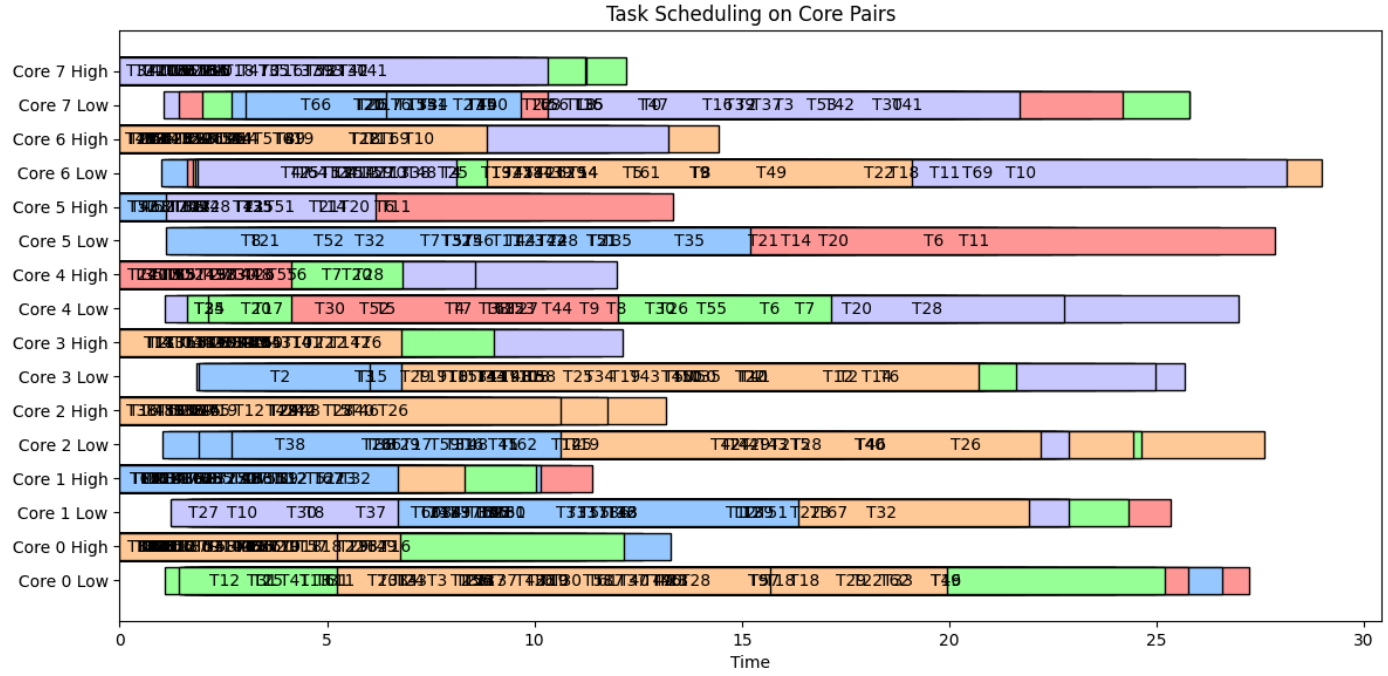


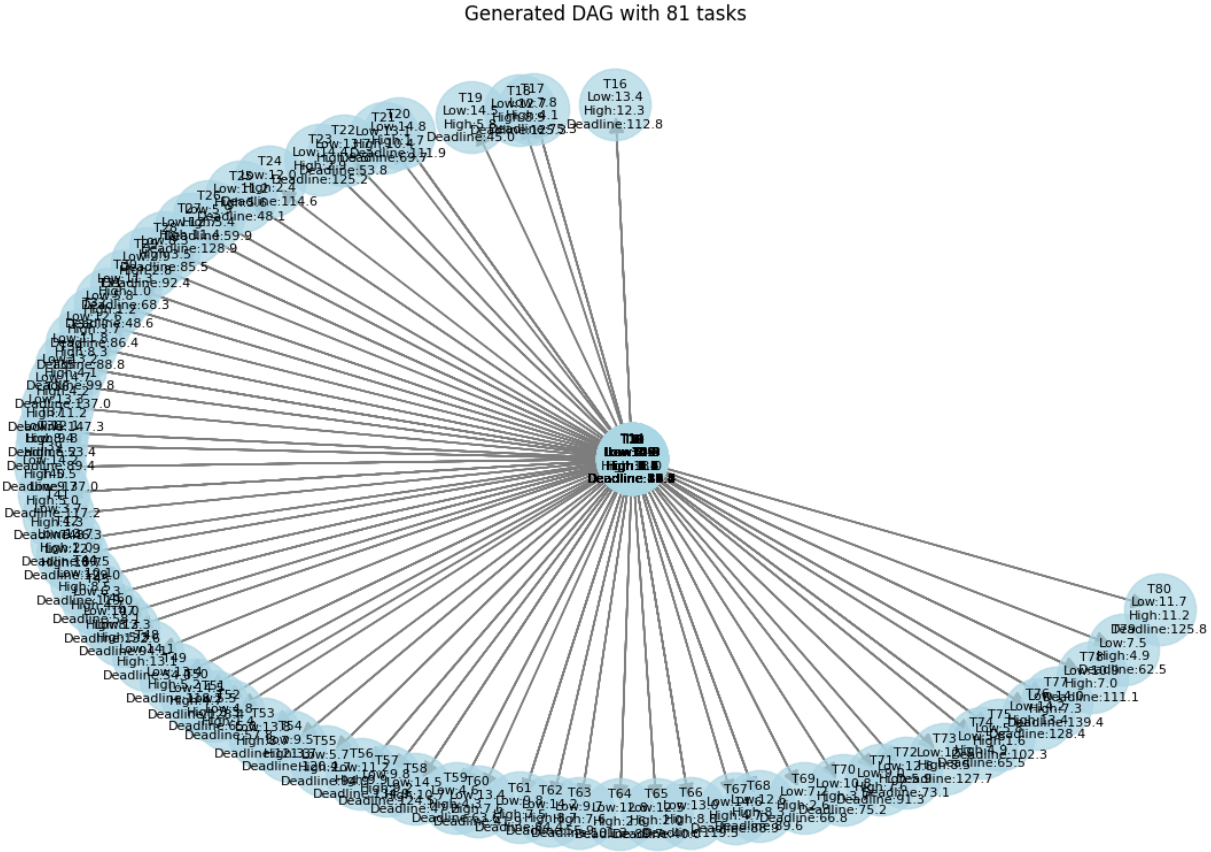
DAG Statistics: num_nodes: 63, num_edges: 246, avg_degree: 3.9047619047619047, critical_path: 1, levels: 2



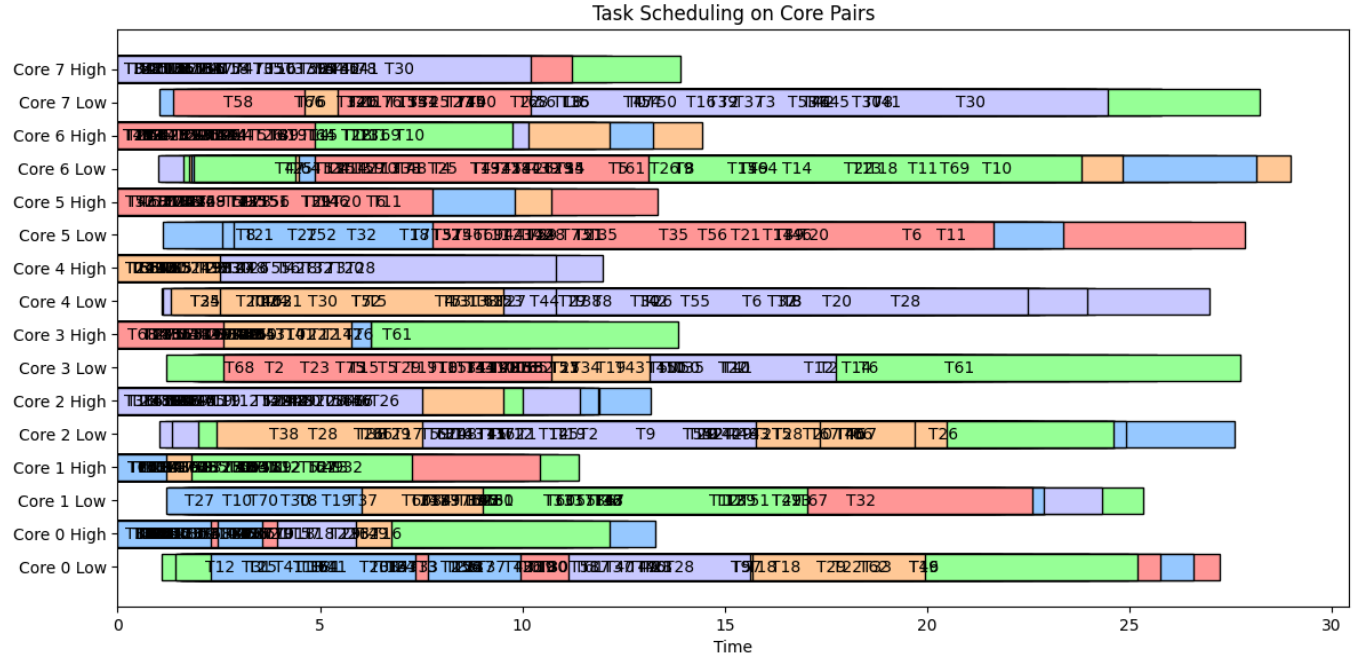


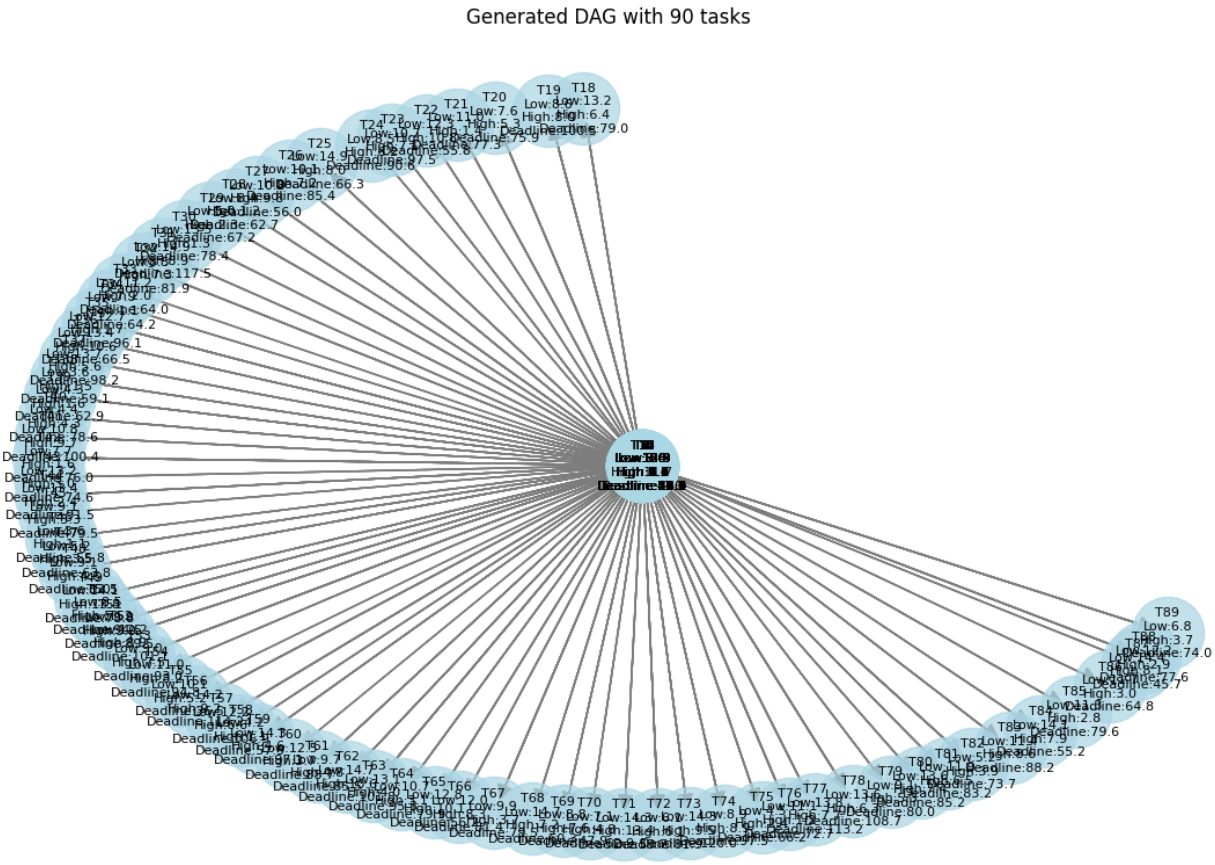
DAG Statistics: num_nodes: 72, num_edges: 331, avg_degree: 4.597222222222222, critical_path: 1, levels: 2



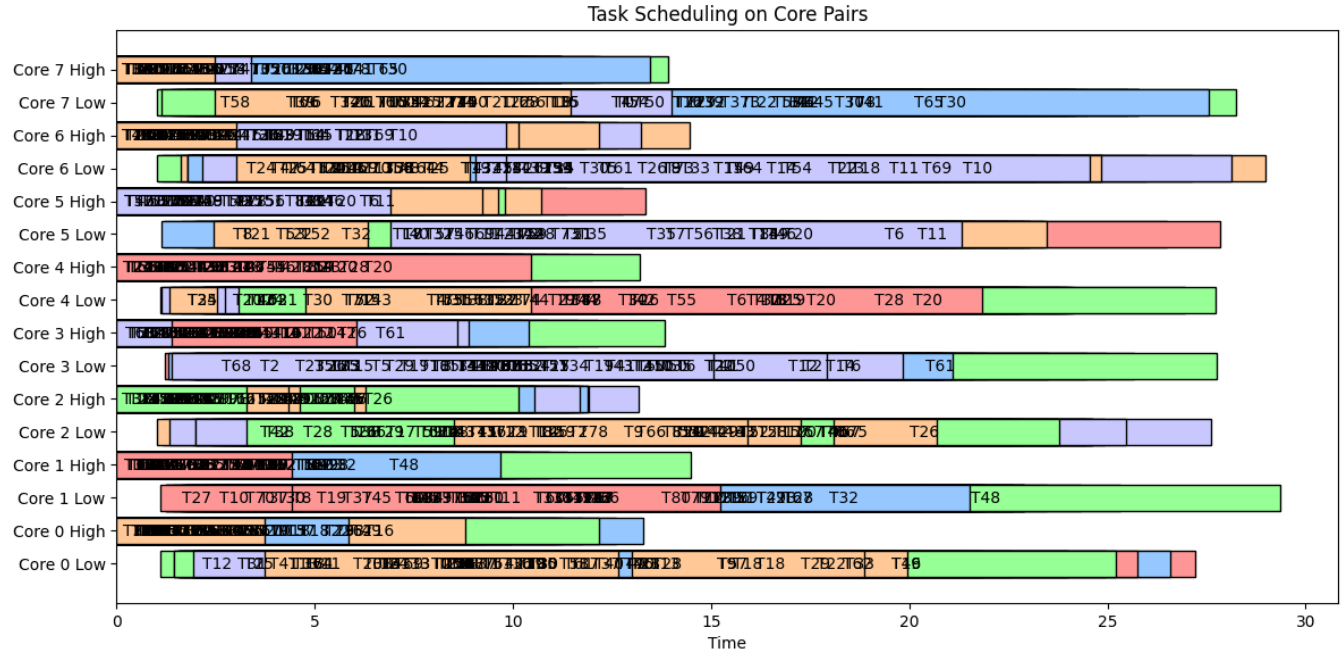


DAG Statistics: num_nodes: 81, num_edges: 425, avg_degree: 5.246913580246914, critical_path: 1, levels: 2

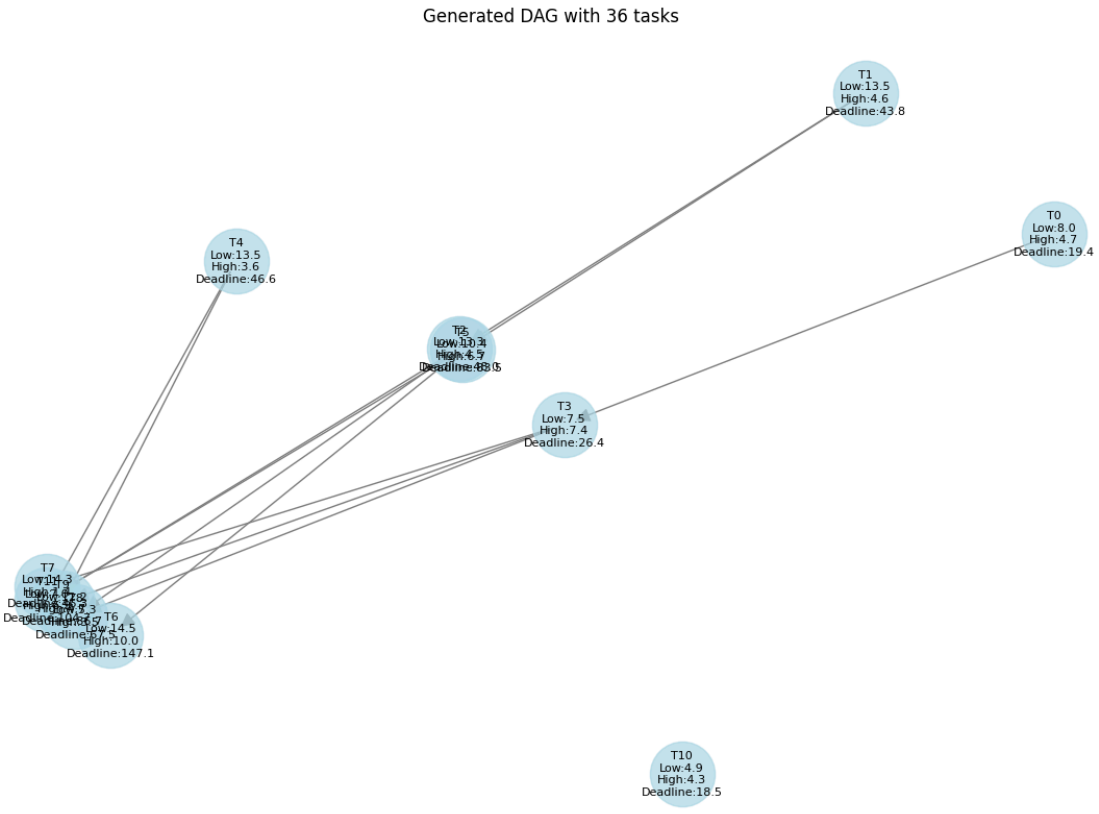




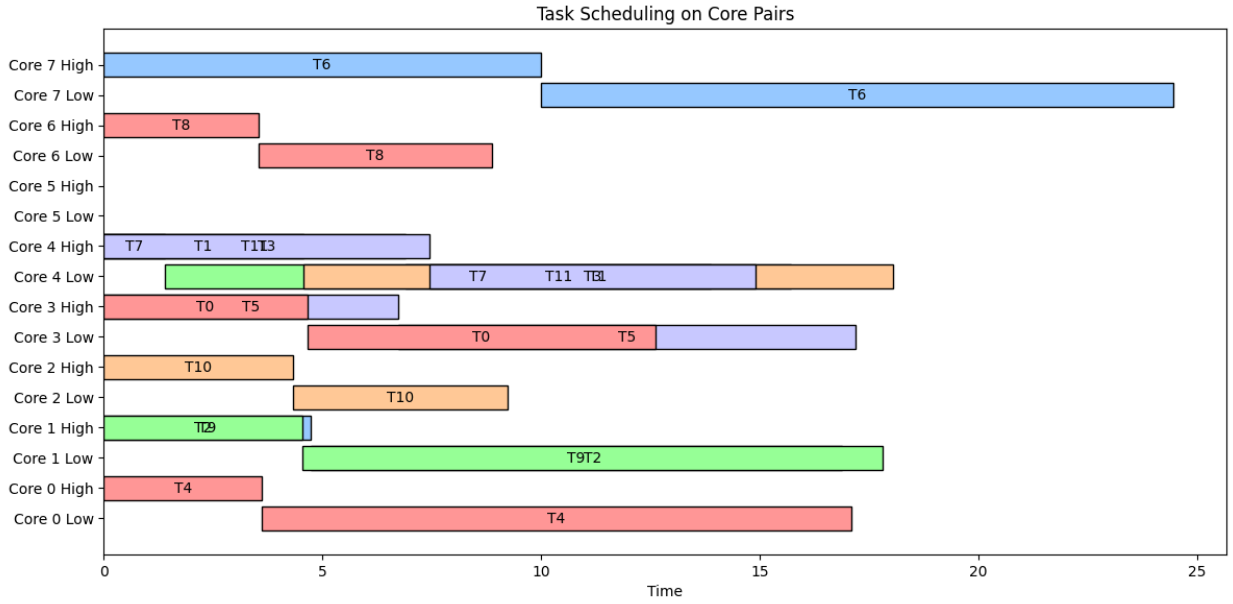
DAG Statistics: num_nodes: 90, num_edges: 514, avg_degree: 5.711111111111111, critical_path: 1, levels: 2

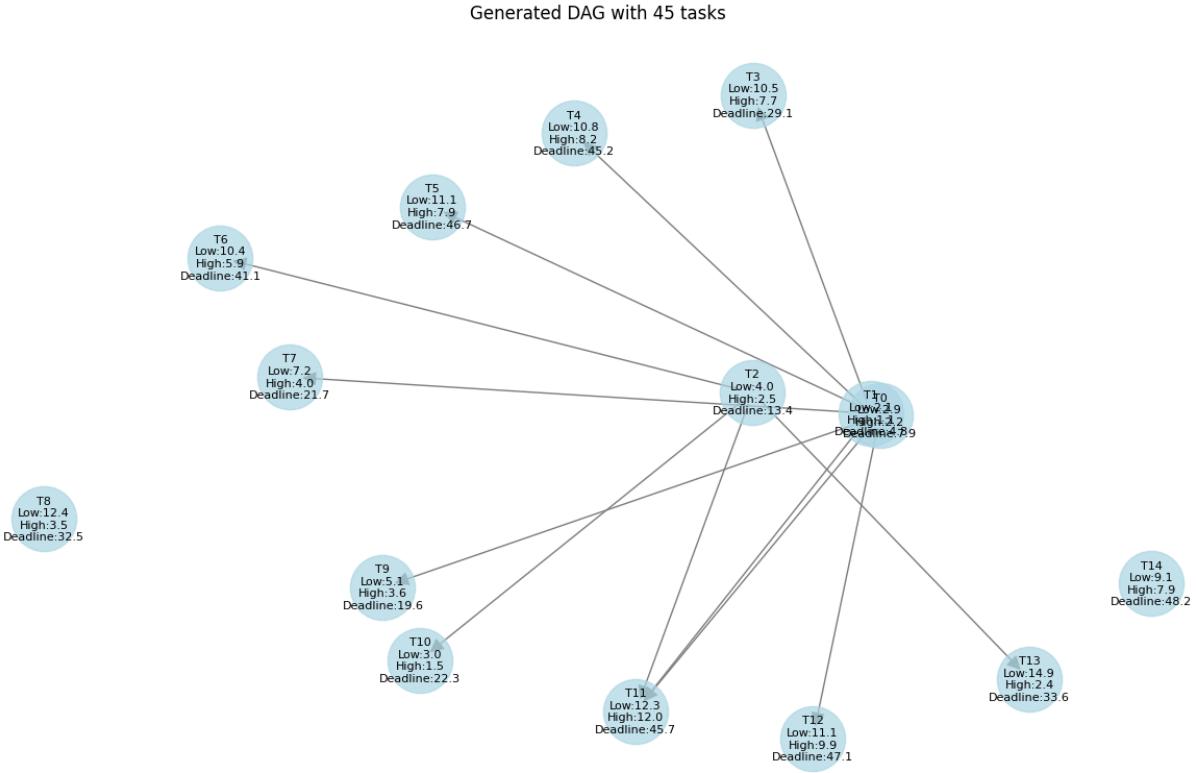


به دلیل تعداد زیاد وظایف و سختی نمایش دقیق زمان‌بندی وظایف روی هسته‌ها با تعداد وظایف کمتر نیز برای زمان‌بندی روی ۸ جفت هسته داریم:

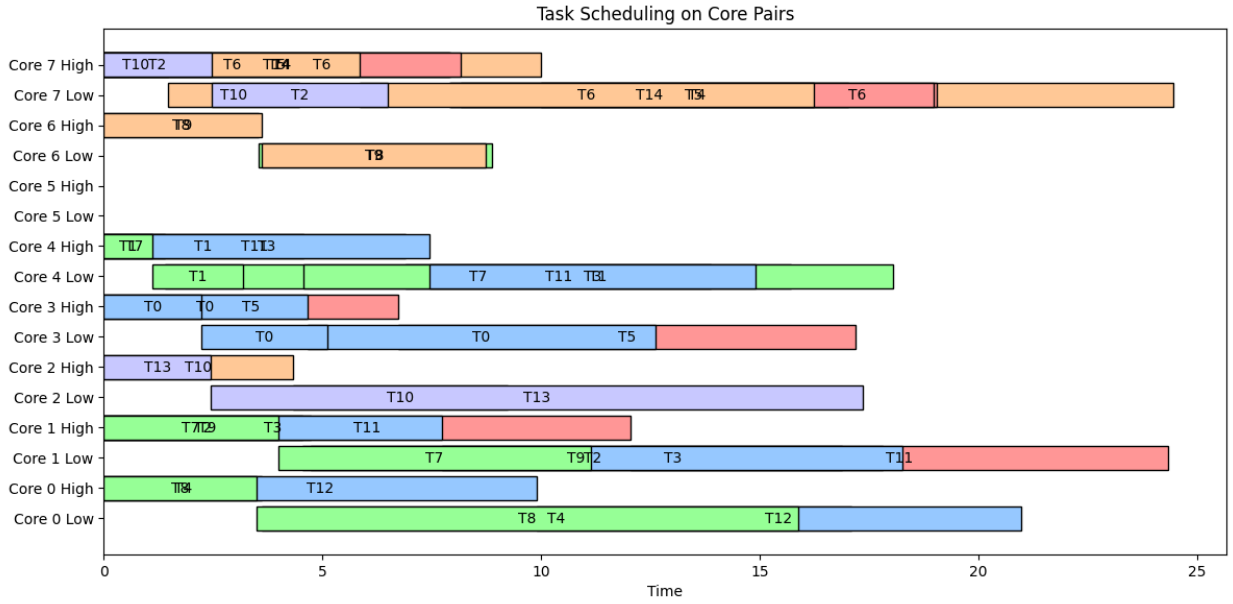


DAG Statistics: num_nodes: 12, num_edges: 12, avg_degree: 1.0, critical_path: 2, levels: 3

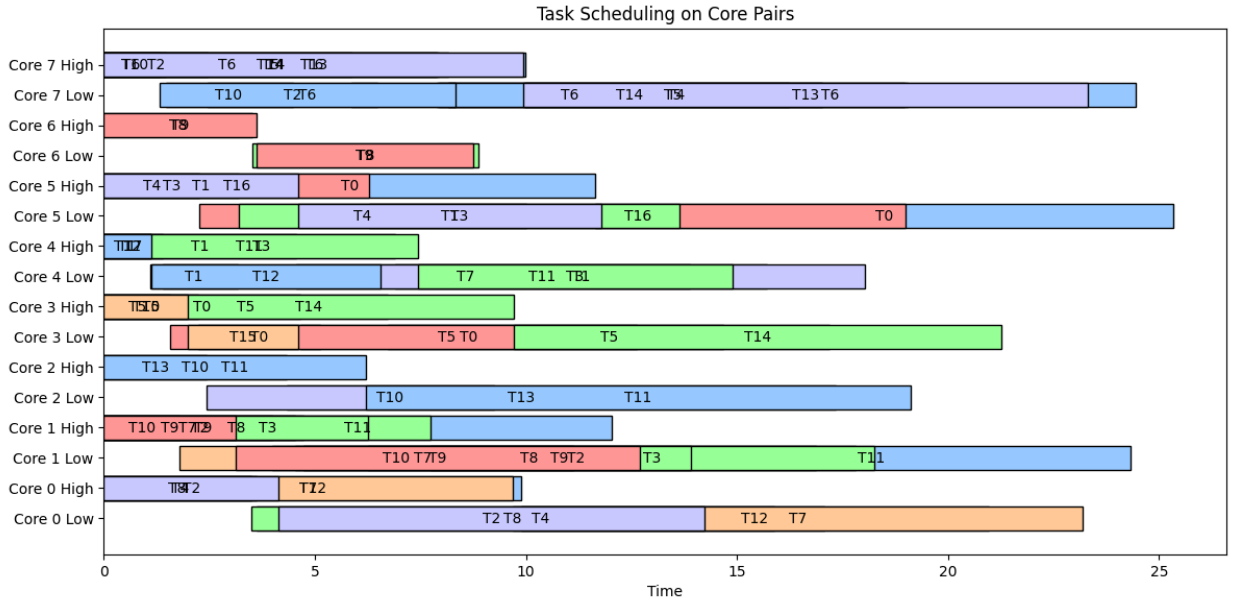


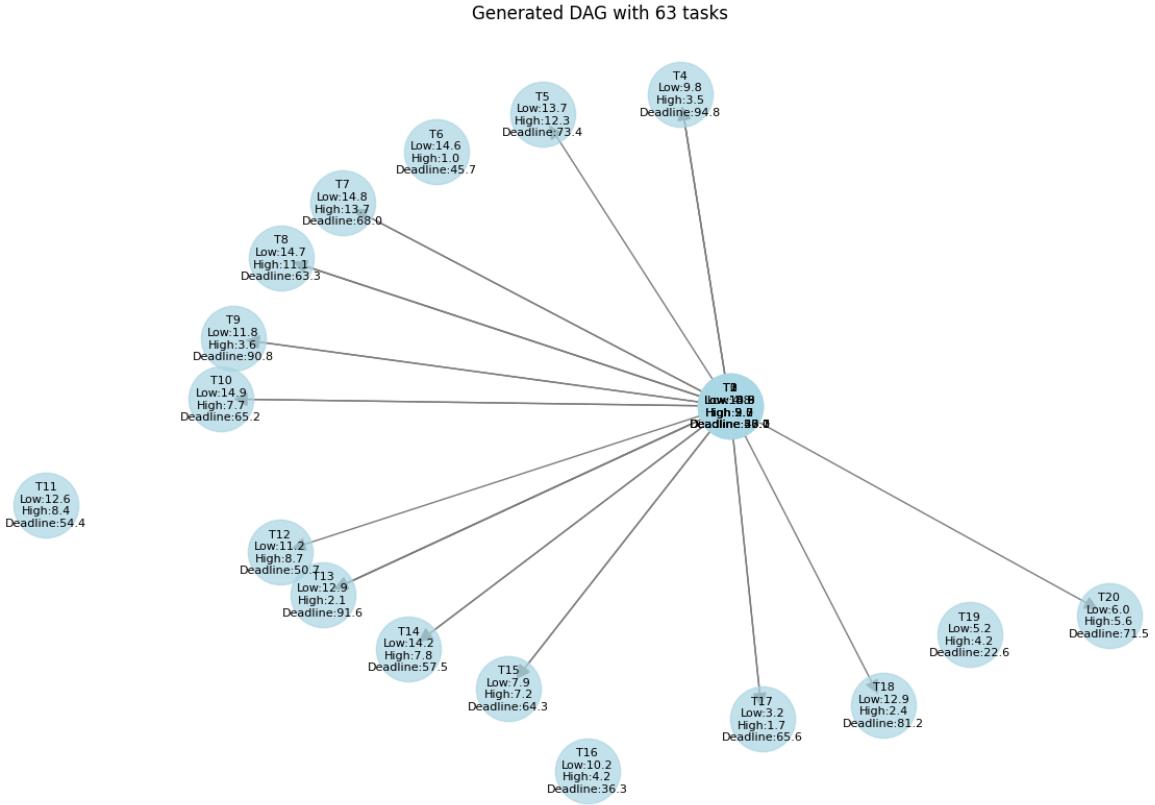


DAG Statistics: num_nodes: 15, num_edges: 12, avg_degree: 0.8, critical_path: 1, levels: 2

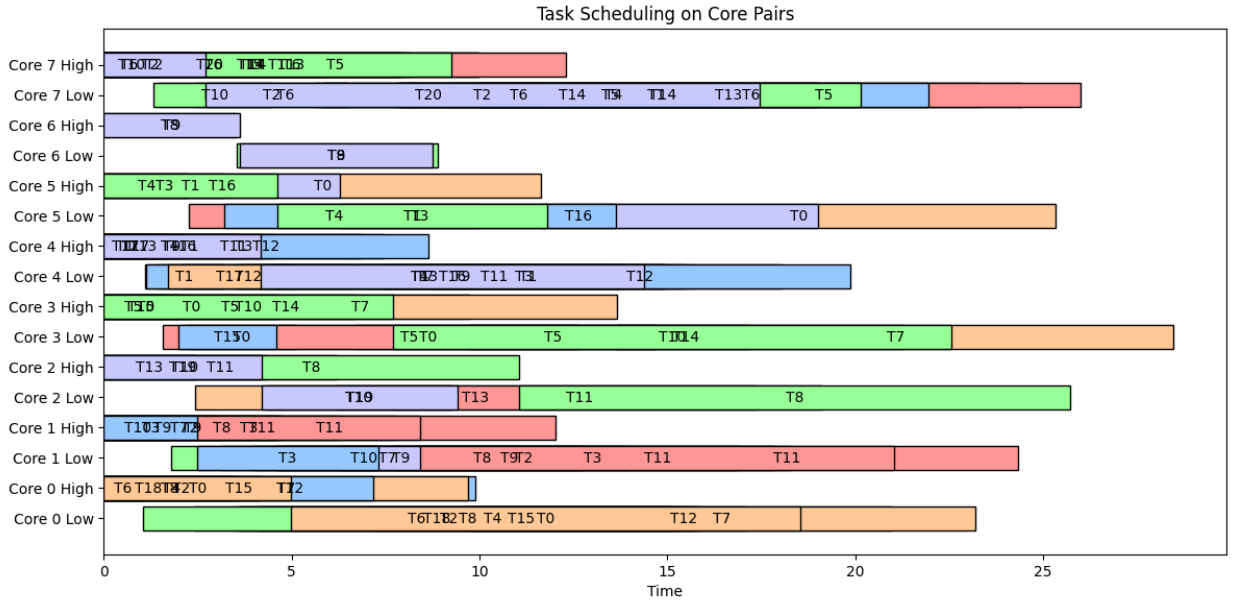


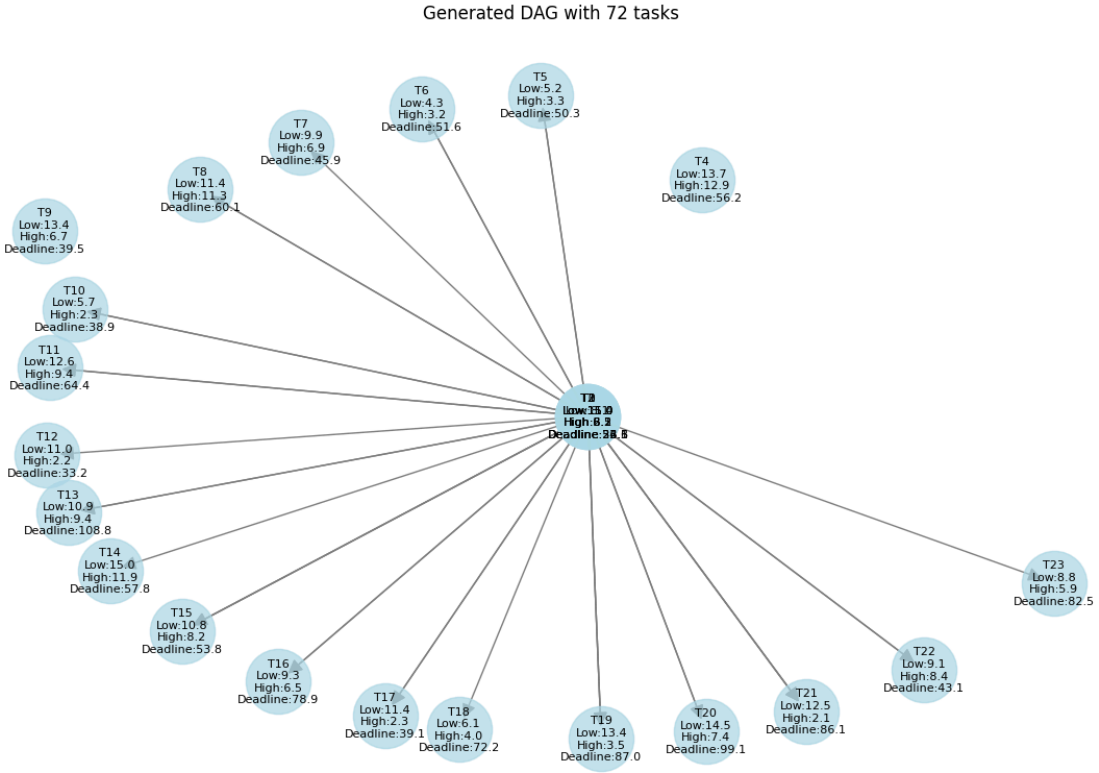
DAG Statistics: num_nodes: 17, num_edges: 15, avg_degree: 0.8823529411764706, critical_path: 1, levels: 2



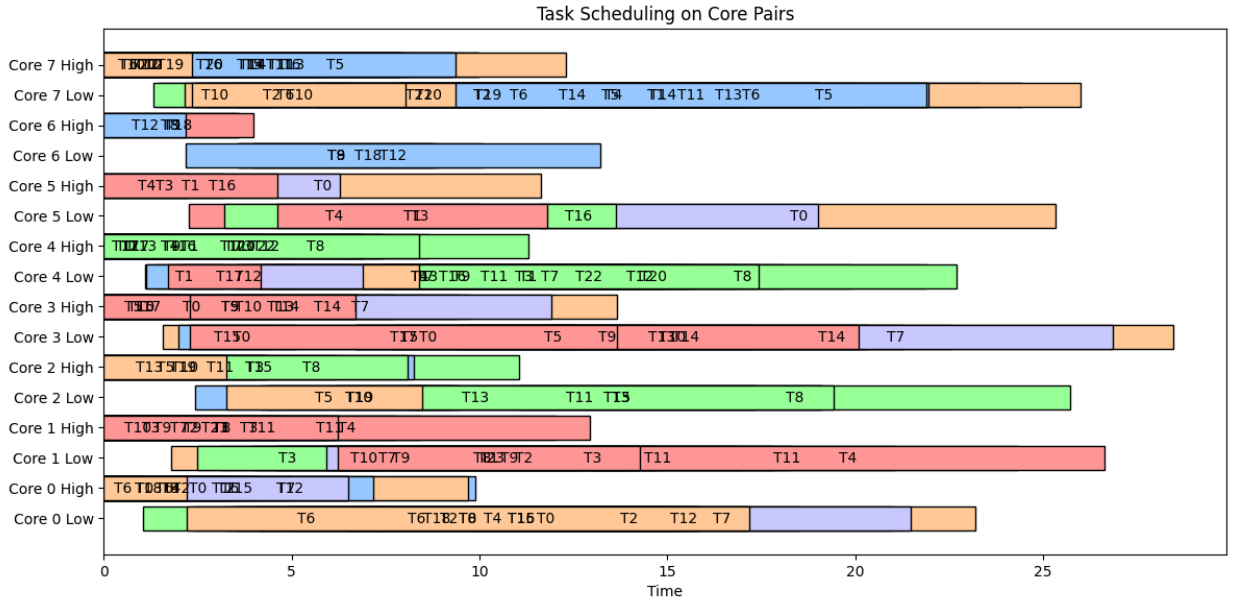


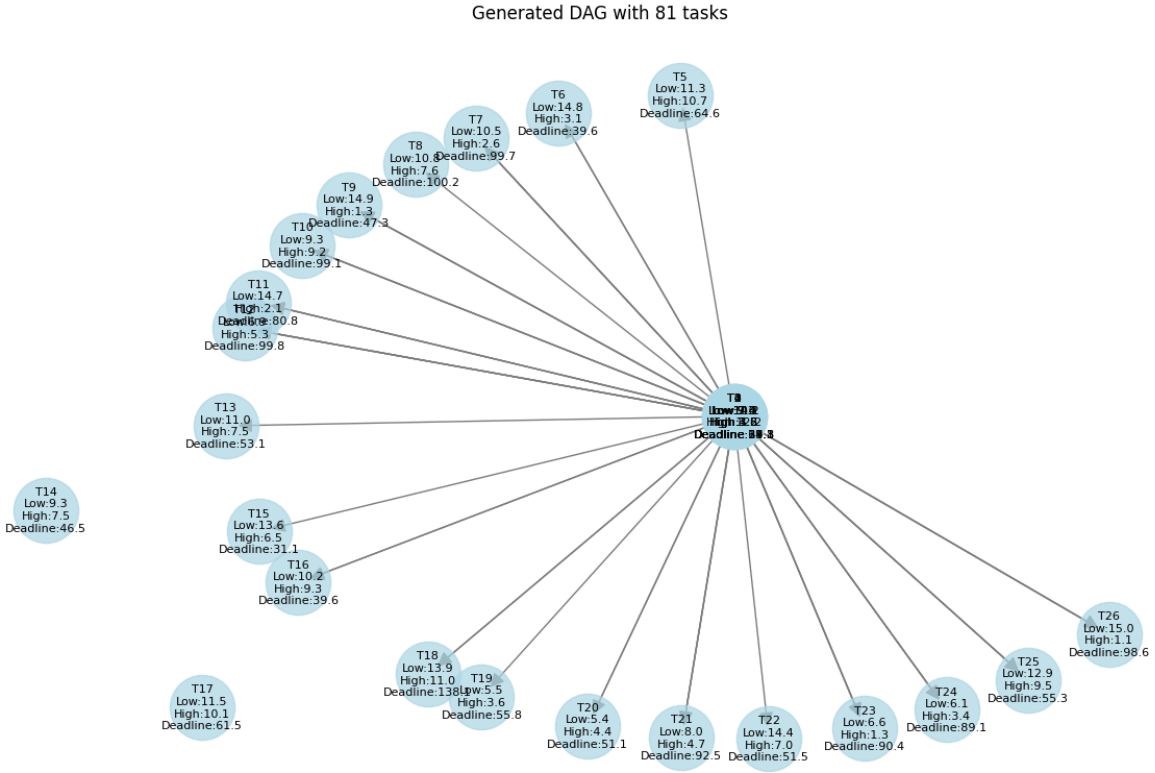
DAG Statistics: num_nodes: 21, num_edges: 26, avg_degree: 1.2380952380952381, critical_path: 1, levels: 2



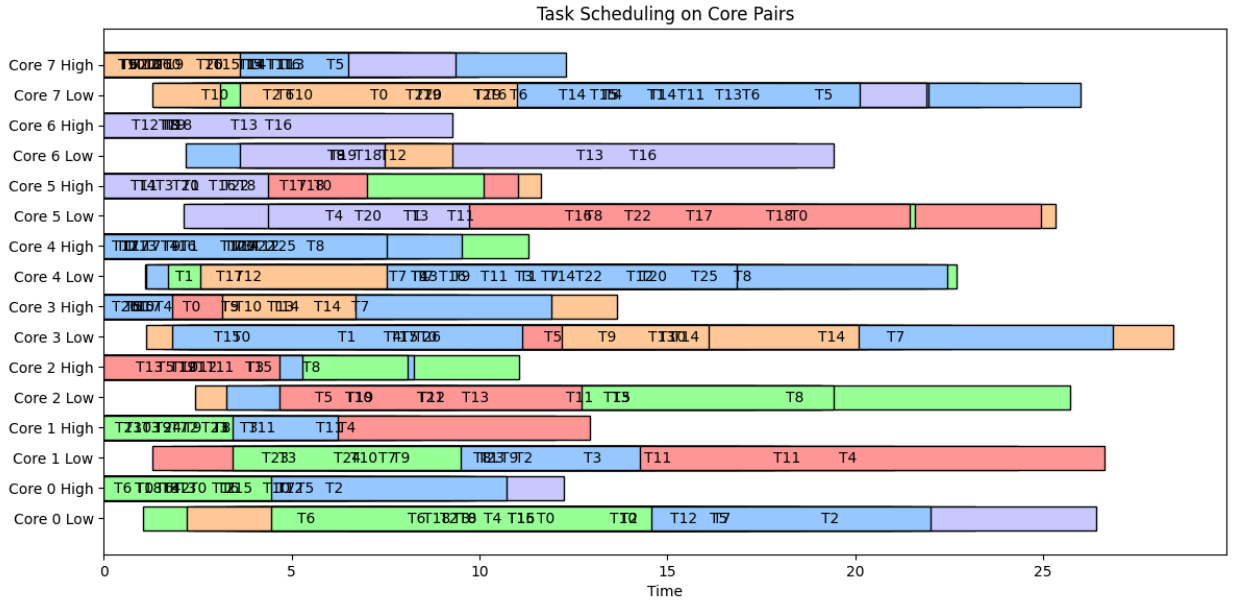


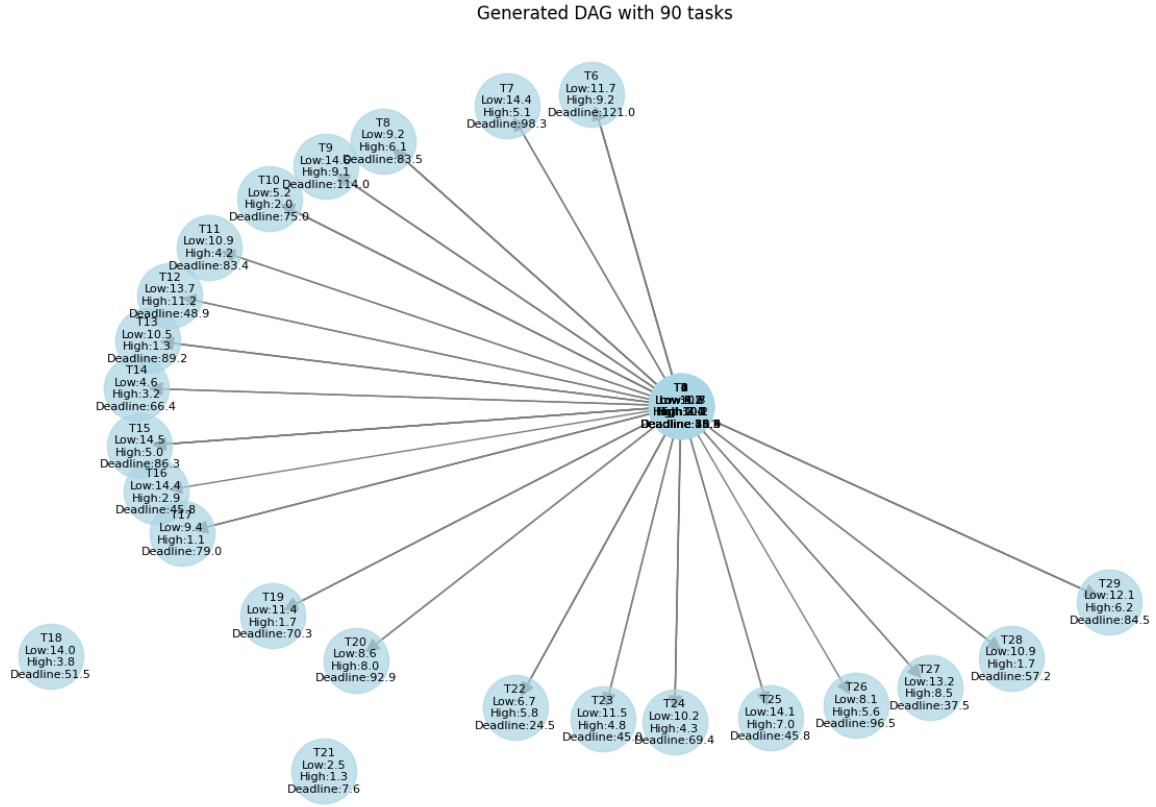
DAG Statistics: num_nodes: 24, num_edges: 34, avg_degree: 1.4166666666666667, critical_path: 1, levels: 2



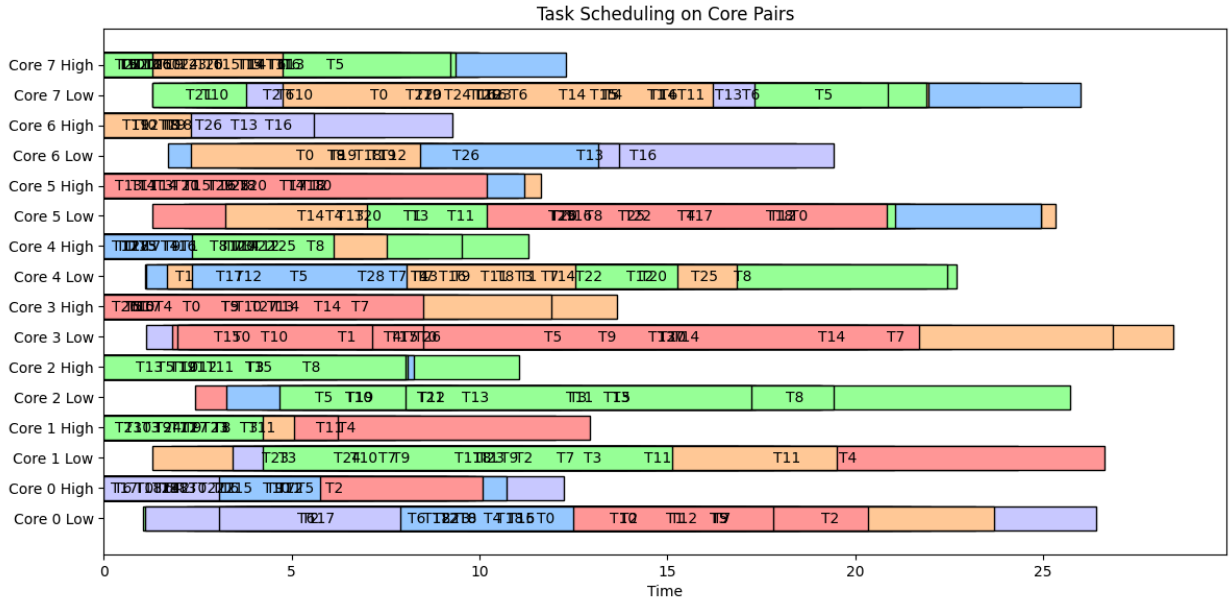


DAG Statistics: num_nodes: 27, num_edges: 45, avg_degree: 1.6666666666666667, critical_path: 1, levels: 2





DAG Statistics: num_nodes: 30, num_edges: 58, avg_degree: 1.9333333333333333, critical_path: 1, levels: 2



در نهایت، ویژگی‌های مثبت تکنیک TASS را بیان می‌کنیم:

1. **بهبود کیفیت سرویس (QoS)**: کیفیت سرویس یکی از معیارهای کلیدی در سیستم‌های نهفته‌ی بی‌درنگ است که نشان‌دهنده‌ی کارایی سیستم در اجرای وظایف به‌موقع و بدون تاخیر است. تکنیک TASS توانسته است کیفیت سرویس را تا **39.78**٪ افزایش دهد (میانگین **18.40**٪).

دلیل این اتفاق این است که TASS از یک رویکرد زمان‌بندی هوشمند استفاده می‌کند که وظایف اصلی را در سریع‌ترین زمان ممکن روی هسته‌های پرقدرت اجرا می‌کند و وظایف پشتیبان تنها در صورت نیاز (یعنی هنگام وقوع خطا) روی هسته‌های کم‌مصرف اجرا می‌شوند. این رویکرد باعث کاهش تأخیر در اجرای وظایف و افزایش بهره‌وری کلی سیستم می‌شود.

★ مزیت: زمانی وظایف سریع‌تر اجرا می‌شوند که سیستم خطا ندارد و در صورت بروز خطا نیز سیستم می‌تواند بدون از دست دادن کیفیت سرویس، وظایف را بازیابی کند.

2. **کاهش مصرف توان پیک**: یکی از چالش‌های اصلی در سیستم‌های چندهسته‌ای، کنترل مصرف توان پیک است که می‌تواند منجر به گرم شدن بیش از حد شود. TASS توانسته است مصرف توان پیک را تا **51.94**٪ در شرایط بدترین حالت و تا **40.21**٪ در سناریوهای واقعی کاهش دهد (میانگین کاهش به ترتیب **34.07**٪ و **28.31**٪).

دلیلش این است که برخلاف روش‌های سنتی که وظایف اصلی و پشتیبان را به‌صورت همزمان اجرا می‌کنند، TASS اجرای وظایف پشتیبان را تا زمانی که نیاز نباشد به تعویق می‌اندازد. این کار باعث می‌شود که هسته‌های اضافی بی‌دلیل فعال نباشند و مصرف توان در لحظات اوج بار کاری کاهش یابد.

★ مزیت: کاهش توان پیک به معنای کاهش هزینه‌های خنک‌کنندگی، افزایش عمر مفید سخت‌افزار و کاهش خطرات ناشی از افزایش دما است.

3. **کاهش دمای سیستم:** دمای بالا می‌تواند باعث کاهش کارایی سیستم و حتی خرابی سخت‌افزار شود. TASS توانسته است دمای سیستم را تا **15.47** درجه سانتی‌گراد کاهش دهد (میانگین **13.60** درجه سانتی‌گراد).

*حال سوال این است که چرا این اتفاق می‌افتد؟

استفاده از محدودیت TSP به جای محدودیت‌های توان سنتی (TDP) باعث می‌شود تا مصرف توان در سطحی نگه داشته شود که از ایجاد نقاط داغ (Hot Spots) جلوگیری شود. علاوه بر این، خاموش کردن هسته‌های پشتیبان غیرضروری در صورت عدم وجود خطا به کاهش دما کمک می‌کند.

★ مزیت: کاهش دما باعث می‌شود سیستم در طولانی‌مدت پایداری بیشتری داشته باشد و نیاز به سیستم‌های خنک‌کنندگی پیچیده و پرهزینه کاهش یابد.

4. **افزایش قابلیت اطمینان:** سیستم‌های نهفته‌ی بی‌درنگ باید حتی در صورت بروز خطاهای سخت‌افزاری یا نرم‌افزاری نیز عملکرد صحیح داشته باشند. TASS توانسته است قابلیت اطمینان سیستم را در برابر هر دو نوع خطا (دائمی و گذرا) افزایش دهد.

به این دلیل این اتفاق می‌افتد که TASS از تکنیک پشتیبان‌گیری سرد (Cold Standby Sparing) استفاده می‌کند، به این معناست که هسته‌های پشتیبان تنها زمانی فعال می‌شوند که خطایی در هسته‌های اصلی رخ دهد. این رویکرد، علاوه بر کاهش مصرف انرژی، باعث افزایش قابلیت اطمینان سیستم می‌شود زیرا احتمال خرابی‌های ناشی از استهلاک مداوم هسته‌های پشتیبان کاهش می‌یابد.

★ مزیت: سیستم می‌تواند بدون وقفه به کار خود ادامه دهد حتی در صورت بروز خطا، در حالی که مصرف انرژی اضافی به حداقل می‌رسد.

5. **مدیریت کارآمد منابع و زمان‌های لختی:** TASS از زمان‌های لختی (Slack Times) به‌صورت بهینه استفاده می‌کند تا هم مصرف انرژی را کاهش دهد و هم کارایی را افزایش دهد.

*چرایی این موضوع:
TASS از تکنیک‌هایی مانند مدیریت پویای ولتاژ و فرکانس (DVFS) و مدیریت پویای توان (DPM) استفاده می‌کند تا زمانی که هسته‌ای بیکار است یا بار کاری سبک دارد، مصرف انرژی آن به حداقل برسد. همچنین، اگر وظیفه‌ی اصلی به درستی اجرا شود، وظیفه‌ی پشتیبان به‌طور کامل حذف می‌شود، که این امر باعث آزاد شدن منابع می‌شود.

★ مزیت: کاهش مصرف انرژی کلی سیستم و افزایش طول عمر باتری در دستگاه‌های قابل حمل.

در مجموع، تکنیک **TASS** یک رویکرد جامع برای بهبود کارایی، کاهش مصرف انرژی، مدیریت حرارتی و افزایش قابلیت اطمینان در سیستم‌های نهفته‌ی بی‌درنگ ارائه می‌دهد. این مزایا باعث می‌شود TASS نسبت به روش‌های پیشین برتری قابل‌توجهی داشته باشد.