به نام خدا

سامانه‌های بی‌درنگ

دکتر سپیده صفری

گزارش نهایی پروژه

پروژه ی شماره 17

زمان بندی در GPU با هدف کاهش Rate Miss Deadline و انرژی مصرفی

زهرا علیپور - 99170529

علی رحیمی - 99170445

بهار ۱۴۰۳

فهرست مطالب

بررسی اجمالی

این پروژه به زمان‌بندی وظایف متناوب با هدف به حداقل رساندن نرخ Miss Deadline و مصرف انرژی در سیستم‌های چند هسته‌ای همگن غیر پیشگیرانه می‌پردازد. این مطالعه شامل توسعه دو الگوریتم، یک الگوریتم همکاری پایه و یک الگوریتم پیشرفته sBEET، برای زمان‌بندی مؤثر وظایف است. الگوریتم sBEET کاهش از دست دادن مهلت و مصرف انرژی از طریق تخصیص بهینه هسته را در اولویت قرار می دهد.

معرفی

زمان‌بندی وظایف در سیستم‌های چند هسته‌ای برای اطمینان از اجرای به موقع و بهره‌وری انرژی بسیار مهم است. این چالش در سیستم‌های غیر پیشگیرانه تشدید می‌شود که در آن‌ها نمی‌توان کارها را پس از شروع قطع کرد. هدف این پروژه ایجاد الگوریتم‌های زمان‌بندی است که با تخصیص دینامیک هسته‌ها به وظایف، از دست دادن مهلت و مصرف انرژی را به حداقل می‌رساند.

مدل سیستم

مفروضات

- زمان اجرای کار را می توان با اختصاص هسته های بیشتر کاهش داد.
- یک هسته را می توان از یک کار قبل از اتمام آن دوباره تخصیص داد.
- حداکثر دو کار را می توان به طور همزمان اجرا کرد.
- تمام هسته ها باید بین دو کار تقسیم شوند یا بیکار بمانند.

الگوریتم ها

الگوریتم uunifast_discard

تابع uunifast_discard یک الگوریتم تولید مجموعه‌های utilization sets برای وظایف در سیستم‌های زمانبندی است. این تابع برای تولید مجموعه‌هایی از درصد استفاده‌ی منابع توسط وظایف استفاده می‌شود که مجموع استفاده‌ی منابع در هر مجموعه برابر با یک مقدار مشخص (مثلاً ۱) باشد. ما از این کد برای تولید یک مجموعه تسک استفاده می‌کنیم. برای اینکه به خوبی تفاوت دو الگوریتم مشخص شود، utilization را برابر 2 قرار داده ایم. از این طریق پریود و سایر اطلاعات تسک را محاسبه می‌کنیم و اطلاعات کلی تسک را در کلاس Task و اطلاعات مربوط به اجرای آن ها را در کلاس TaskProfile قرار می‌دهیم.

الگوریتم Cooperative

حال به بررسی پیاده سازی الگوریتم Cooperative می پردازیم. الگوریتم Cooperative یک روش زمان‌بندی کار ساده است که در آن وظایف به صورت مشارکتی اجرا می‌شوند. در اینجا روند دقیق این الگوریتم را توضیح میدهیم:

1. Task Queueing: ابتدا وظایف در صف اولویت بر اساس پریودشان قرار می‌گیرند. اینکار با استفاده از heapq انجام میشود.

2. Scheduling: در متد schedule، زمان‌بندی از زمان 0 شروع می‌شود و تا پایان هایپر پریود ادامه می‌یابد (در عمل به جای هایپر پریود مقدار کوچکتری قرار داده شده تا زمان اجرا برنامه خیلی زیاد نشود).
   در هر تکرار از حلقه while، وظیفه‌ای که پریود کمتری دارد از صف خارج می‌شود. این وظیفه انتخاب شده توسط متد execute_task اجرا می‌شود.

3. Execution and Timing: در این الگوریتم، همه 6 کور پردازنده را به همان تسک در حال اجرا اختصاص می‌دهیم. سپس به اندازه یک واحد زمانی (که در کد معادل 0.1 است) جلو می‌رویم. زمان سپری شده به current_t اضافه می‌شود و از زمان اجرای تسک کاسته می‌شود. اطلاعات زمان‌بندی در plot_info ذخیره می‌گردد تا در زمان رسم نمودار زمان بندی استفاده شود. اگر زمان فعلی از ددلاین بیشتر باشد، پیام "Deadline missed" چاپ می شود. اگر زمان اجرای تسک تمام شده باشد، پیام executed چاپ می شود. بعد از اجرای وظیفه، زمان ورود بعدی آن به‌روزرسانی می‌شود و مجدداً به صف اضافه می‌شود.

الگوریتم sBEET

الگوریتم sBEET یک روش زمان‌بندی پیچیده‌تر است که اجرای کار را بر اساس در دسترس بودن هسته‌های پردازنده بهینه می‌کند. در اینجا روند دقیق این الگوریتم را توضیح میدهیم:

1. Initialization: در ابتدا avaliable_cores را برابر 6 قرار می دهیم و در حلقه قرار می‌گیریم و یک واحد زمانی را شروع می‌کنیم.

2. Task Selection: سپس بهترین جفت وظایف با توجه به تعداد هسته‌های موجود انتخاب می‌شوند. الویت وظایف در اینجا مشابه الگوریتم Rate Monotonic است. اگر همه هسته ها در دسترس باشند، دو وظیفه بالای صف را برمی‌دارد و تمام حالت های اجرای آن را بررسی می‌کند. سپس از بین آن ها بهینه ترین را انتخاب می‌کند. اگر هم فقط بخشی از هسته ها در دسترس باشند، بین تسک در حال اجرا و تسک دیگر بالای صف تقسیم میکند(هسته های تسک در حال اجرا کمتر نمیشود).

3. Execution: سپس در متد execute_task عملیات اختصاص دادن کورها و افزودن جزییات plot_info و تغییر حالت تسک به حالت اجرا انجام می شود.

در مرحله بعد در تابع time_passage شبیه‌سازی گذر زمان انجام می‌شود و زمان فعلی و زمان اجرای وظایف به‌روزرسانی می‌شود.

4. Finalization: در متد finalize_task بررسی می شود که آیا زمان اجرا تسک تمام شده، ددلاین میس/میت شده یا نه و با توجه به شرایط، هسته ها بازگردانده می شود و اطلاعات تسک آپدیت می شود.

متدولوژی

پیاده سازی کد

این پروژه با استفاده از زبان پایتون و کتابخانه هایی که در آن تعریف شده بود پیاده سازی شد. اجزای کلیدی این پروژه عبارتند از:

Task Generation:

- کلاس TaskGen با استفاده از الگوریتم UUniFast وظایفی را بر اساس مجموعه های استفاده تولید می کند.
- پروفایل های وظیفه از فایل های CSV استخراج و پیوست می شوند.

الگوریتم های زمان بندی:

- CooperativeScheduler:
  - وظایف را به ترتیب اجرا می کند، تخصیص هسته را به روز می کند و زمان های اجرا را ردیابی می کند.
- SBEETScheduler:
  - تمام تخصیص های اصلی ممکن را برای کار با بالاترین اولویت ارزیابی می کند.
  - برنامه ای را انتخاب می کند که به حداقل رساندن مهلت زمانی و مصرف انرژی را کاهش می دهد.

Plotting:

تابع plot_tasks جدول زمانی اجرای کار را برای هر دو الگوریتم تجسم می کند.

محاسبه هایپر پریود

دوره بیش از حد مجموعه وظایف به عنوان کمترین مضرب مشترک (LCM) دوره های تکالیف محاسبه می شود، و اطمینان حاصل می کند که همه برنامه های کار به طور قابل پیش بینی تکرار می شوند.

ورودی‌ها

Task Profile: شامل زمان اجرا، توان مصرفی و انرژی مصرفی هر تسک بر اساس تعداد هسته‌های اختصاص داده شده است.
Task Set: شامل پریود و بهره‌وری هر تسک است.

خروجی‌ها

نمودار زمان‌بندی الگوریتم‌ها
نمودار توان مصرفی الگوریتم‌ها
نمودار مقایسه نرخ Miss Deadline با بهره‌وری
نمودار مقایسه انرژی مصرفی با بیشترین بهره‌وری
نمودار سرعت (Speedup) نسبت به الگوریتم Cooperative با افزایش بهره‌وری

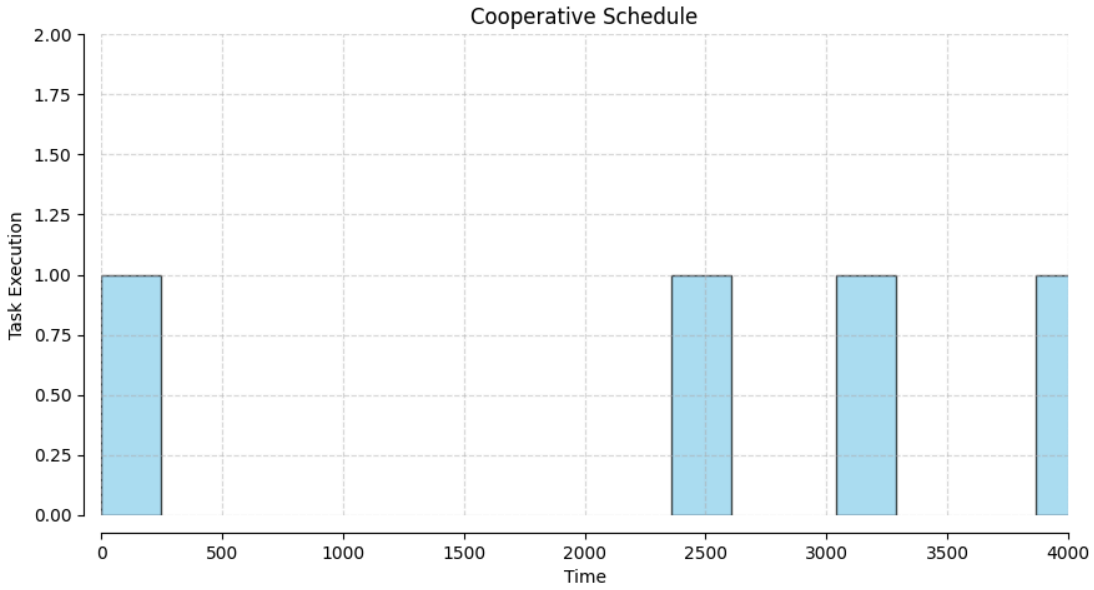
تحلیل داده‌ها

برای بررسی بخش‌های مختلف پاسخ به صورت‌های گوناگون عمل کرده‌ایم. برای بخش‌های نمودار زمان‌بندی الگوریتم‌ها و نمودار توان الگوریتم‌ها، هر الگوریتم را روی یک تسک ست یکسان اجرا کردیم و خروجی‌ها را بررسی کردیم. اما برای بررسی نمودار مقایسه Deadline Miss rate با افزایش Utilization، نمودار مقایسه انرژی مصرفی با افزایش Utilization و نمودار Speedup نسبت به الگوریتم Cooperative با افزایش Utilization، کد را بر روی تسک ست‌های مختلف با utilizatioin های مختلف ران کرده و نتایج را بررسی کرده‌ایم.

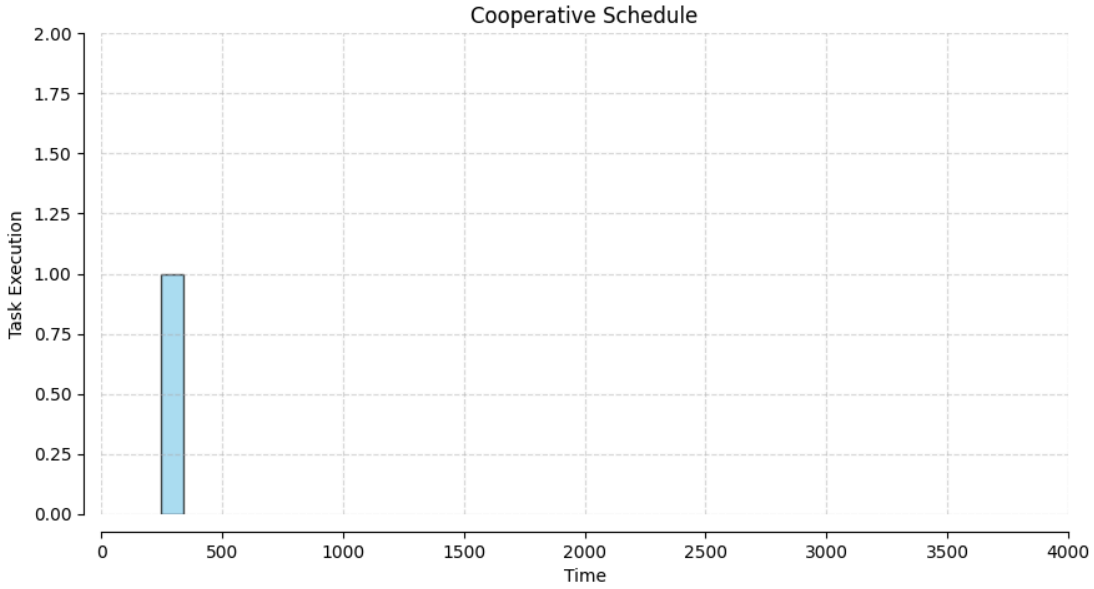
1. نمودار زمان‌بندی الگوریتم‌ها

این نمودارها نشان‌دهنده ترتیب اجرای تسک‌ها و تخصیص هسته‌های پردازنده در طول زمان برای دو الگوریتم Cooperative و sBEET هستند.
نمودار زمان‌بندی الگوریتم Cooperative نشان می‌دهد که تمام هسته‌ها به صورت ثابت به یک تسک اختصاص داده می‌شوند و تا اتمام تسک تغییر نمی‌کنند. این باعث می‌شود که تسک‌ها به ترتیب اجرا شوند و هیچ بهینه‌سازی خاصی در تخصیص هسته‌ها وجود نداشته باشد. از طرف دیگر، نمودار زمان‌بندی الگوریتم sBEET نشان می‌دهد که هسته‌ها به صورت داینامیک بین تسک‌ها توزیع می‌شوند. این تخصیص داینامیک باعث می‌شود که تسک‌ها سریع‌تر اجرا شده و هسته‌ها به صورت بهینه‌تری استفاده شوند.

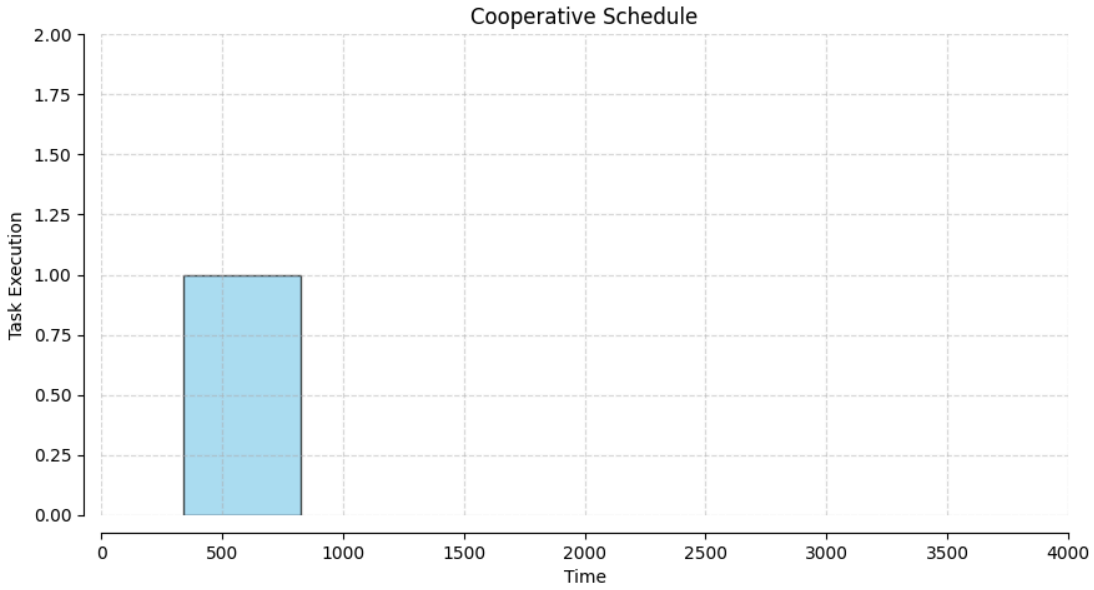
bfs:


Cooperative Schedule
Task Execution
Time
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
500
1000
1500
2000
2500
3000
3500
4000

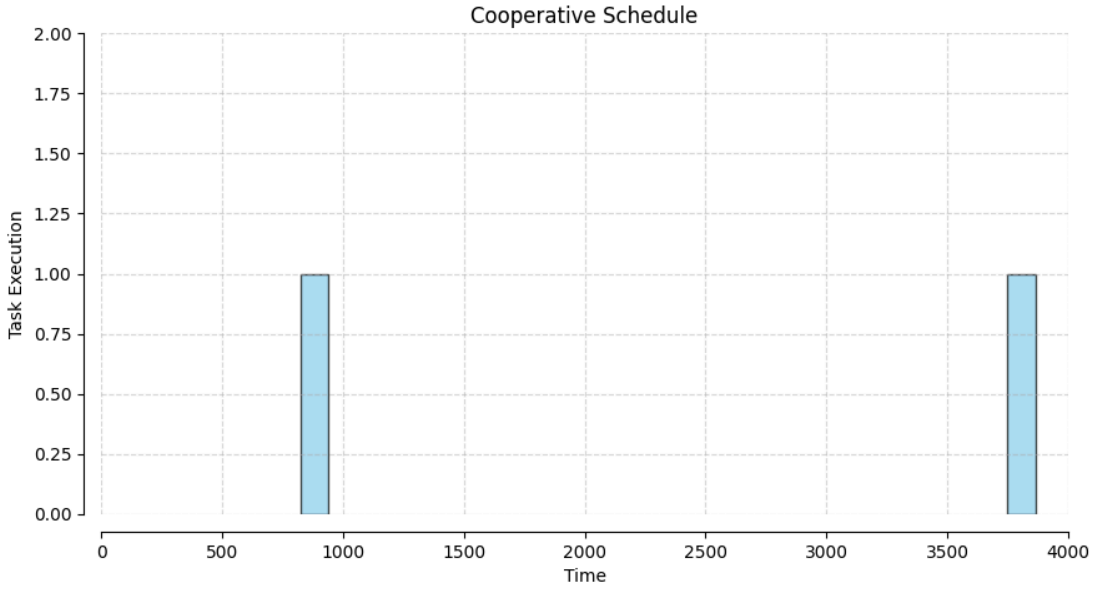
dxtc:


Cooperative Schedule
Task Execution
Time
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
500
1000
1500
2000
2500
3000
3500
4000

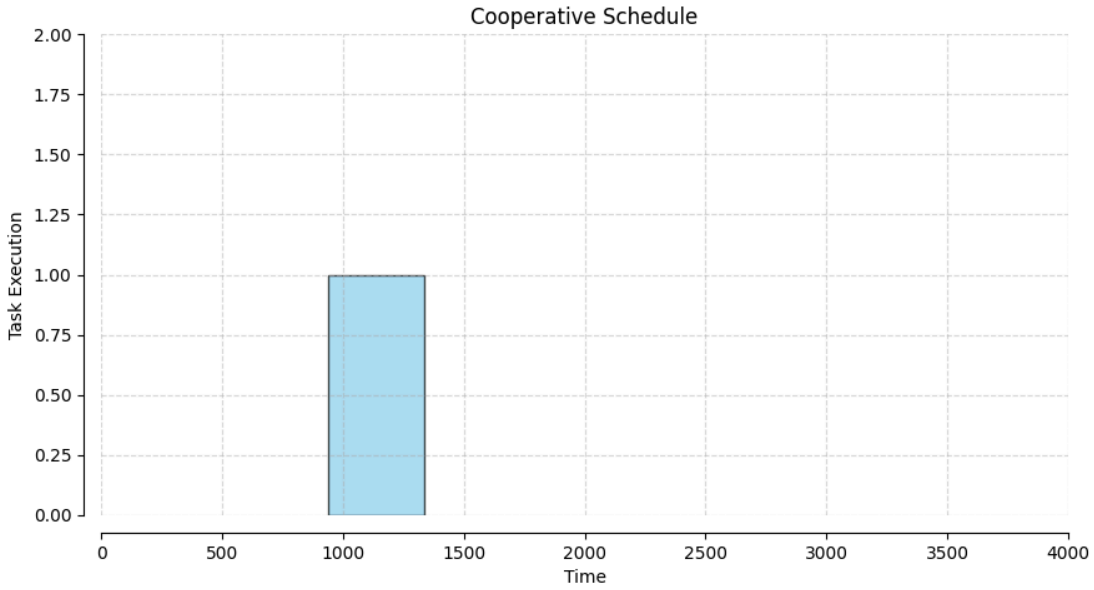
hist:


Cooperative Schedule
Task Execution
Time
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
500
1000
1500
2000
2500
3000
3500
4000

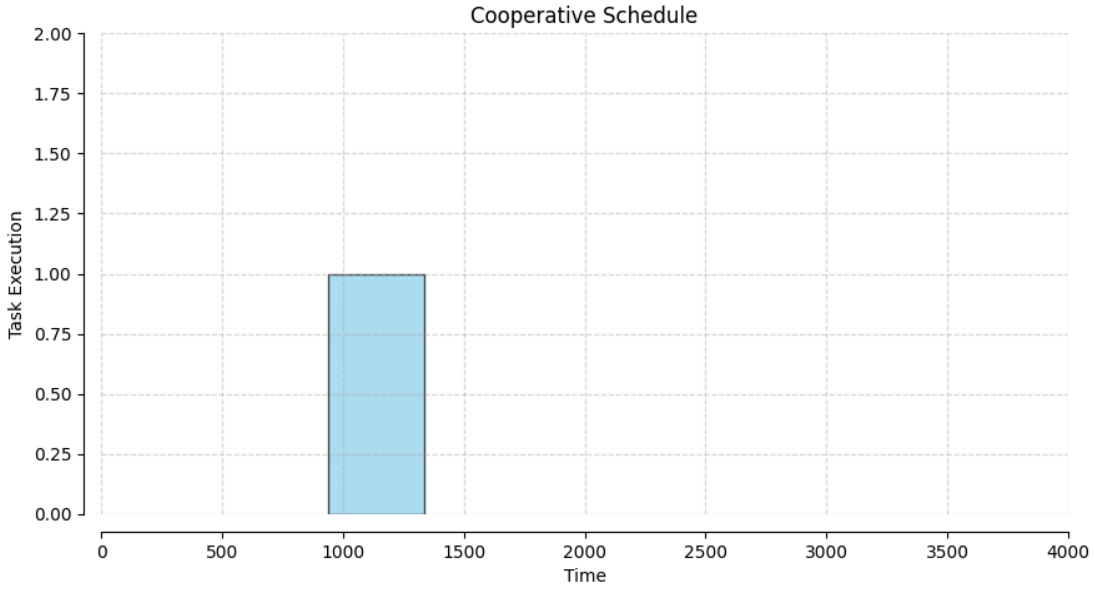
hist2:


Cooperative Schedule
Task Execution
Time
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
500
1000
1500
2000
2500
3000
3500
4000

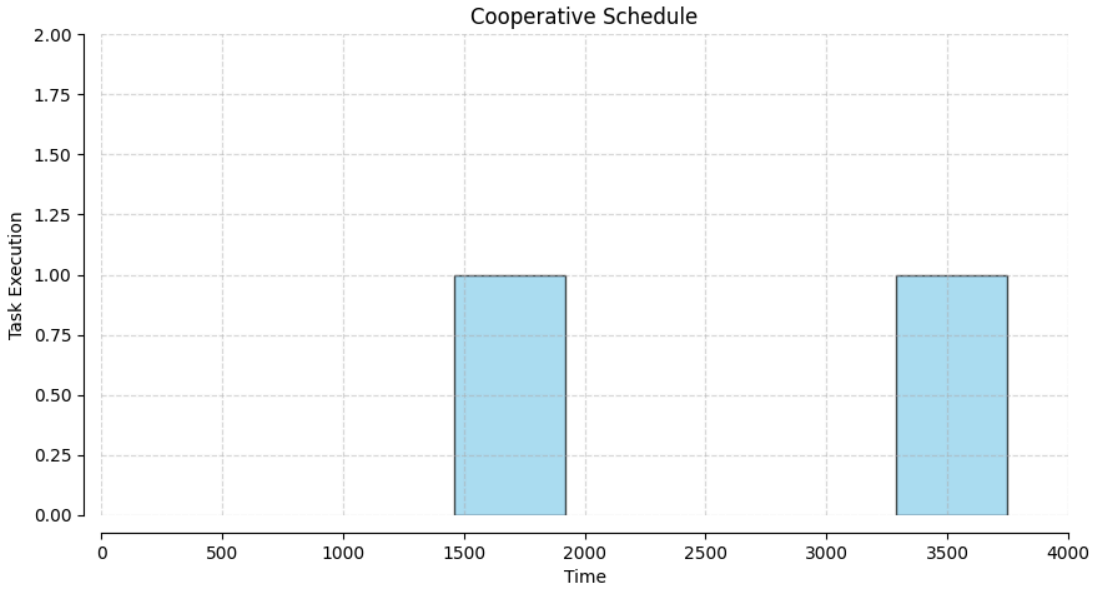
hotspot:



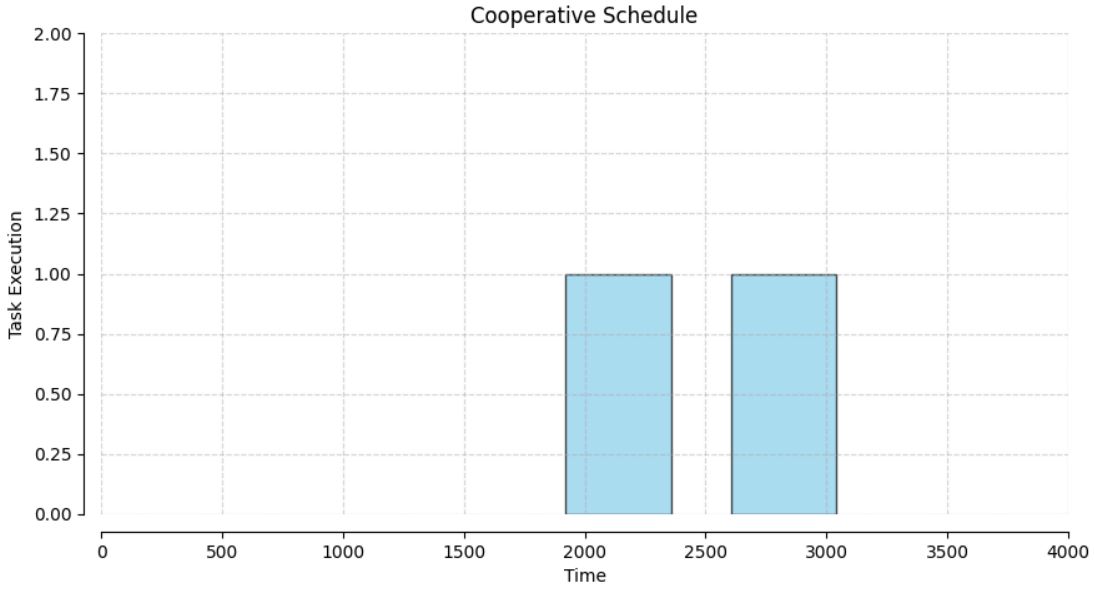
mmul:



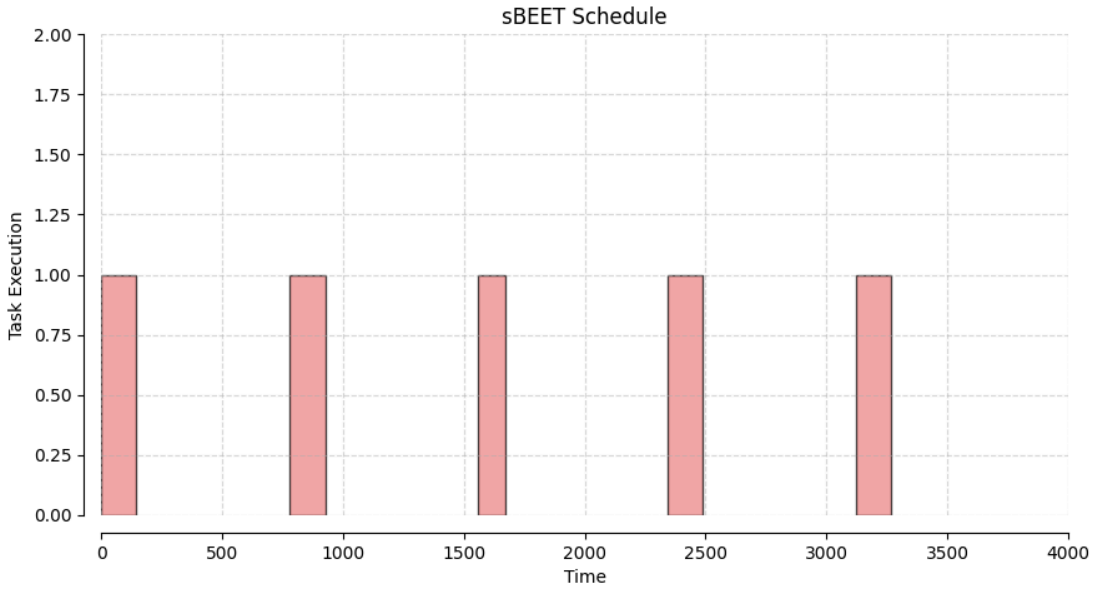
mmul2:



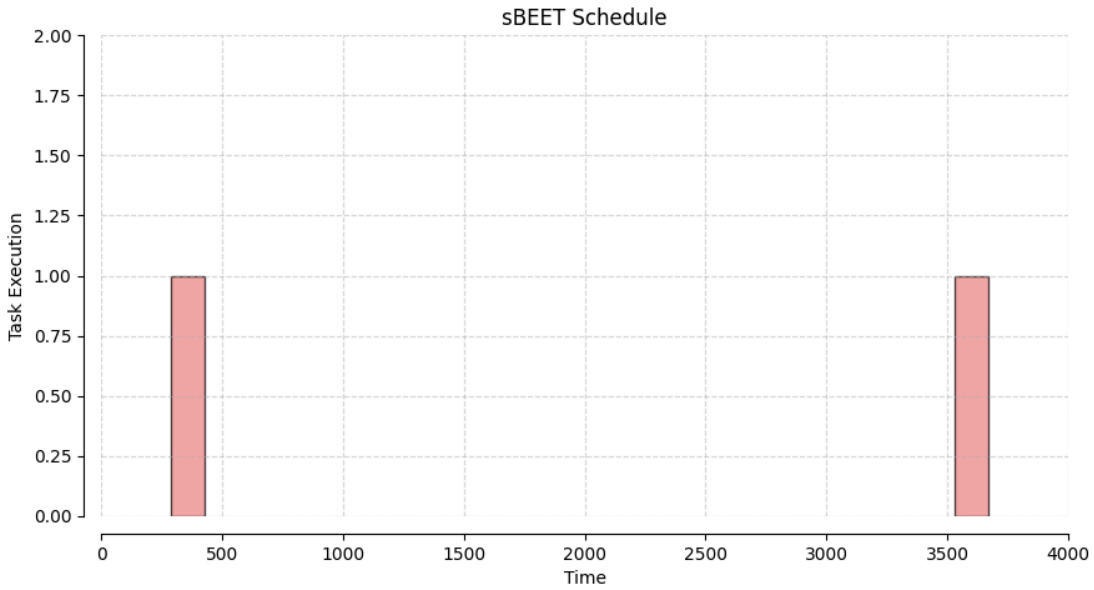
stereodisparity:



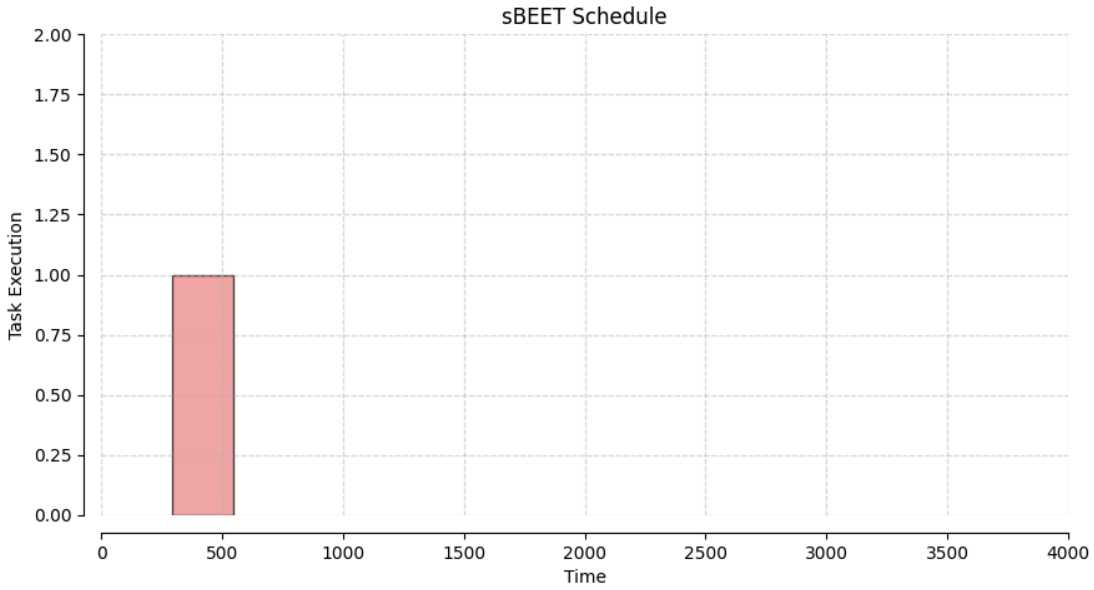
Bfs:


sBEET Schedule
Task Execution
Time
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
500
1000
1500
2000
2500
3000
3500
4000

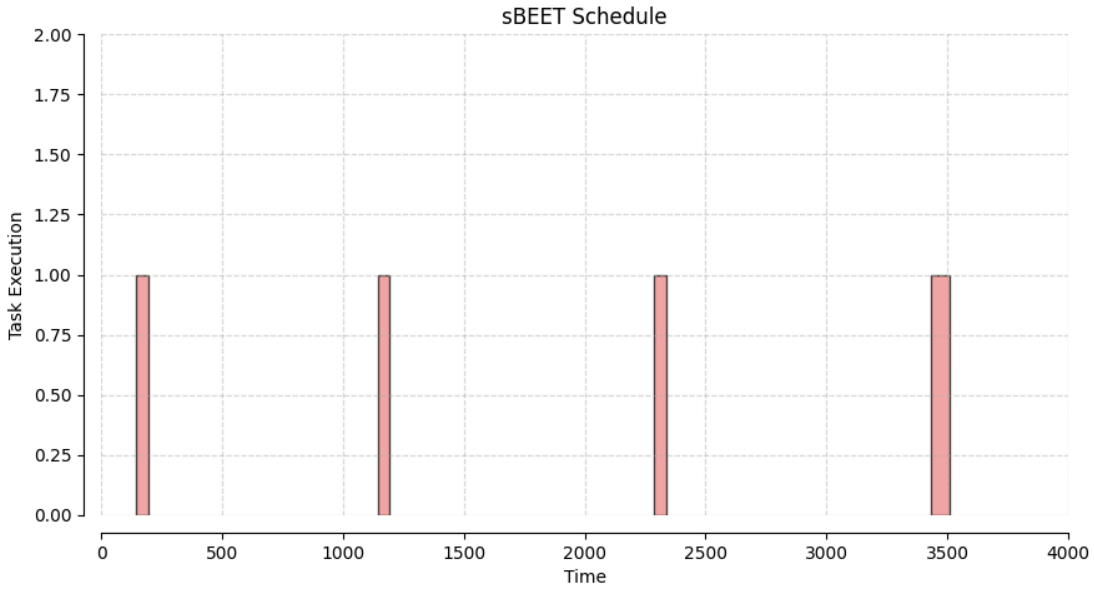
Dxtc:


sBEET Schedule
Task Execution
Time
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
500
1000
1500
2000
2500
3000
3500
4000

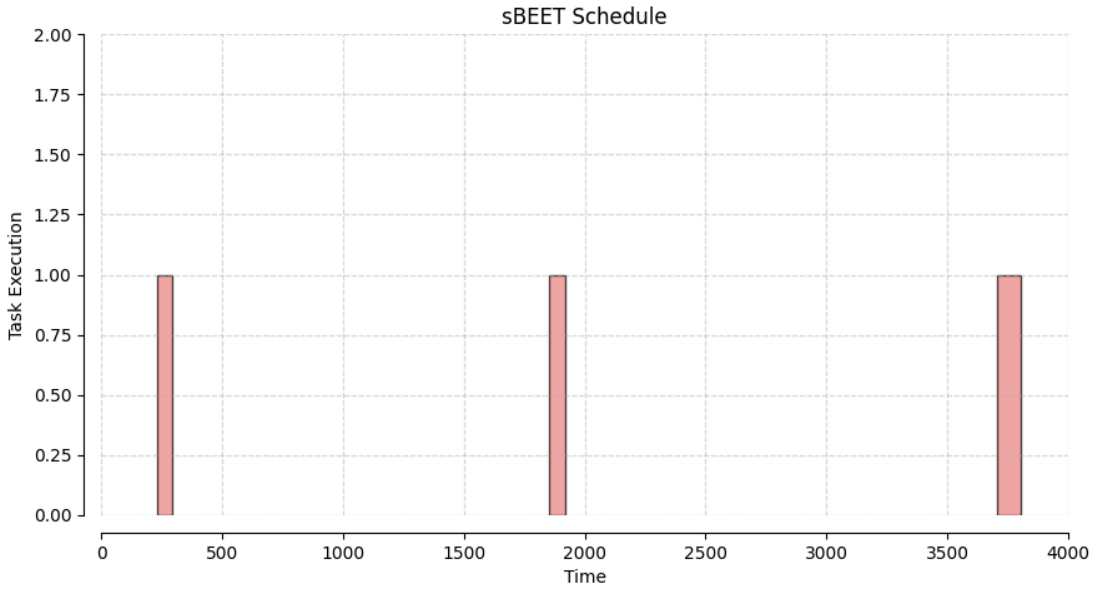
Hist:


sBEET Schedule
Task Execution
Time
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
500
1000
1500
2000
2500
3000
3500
4000

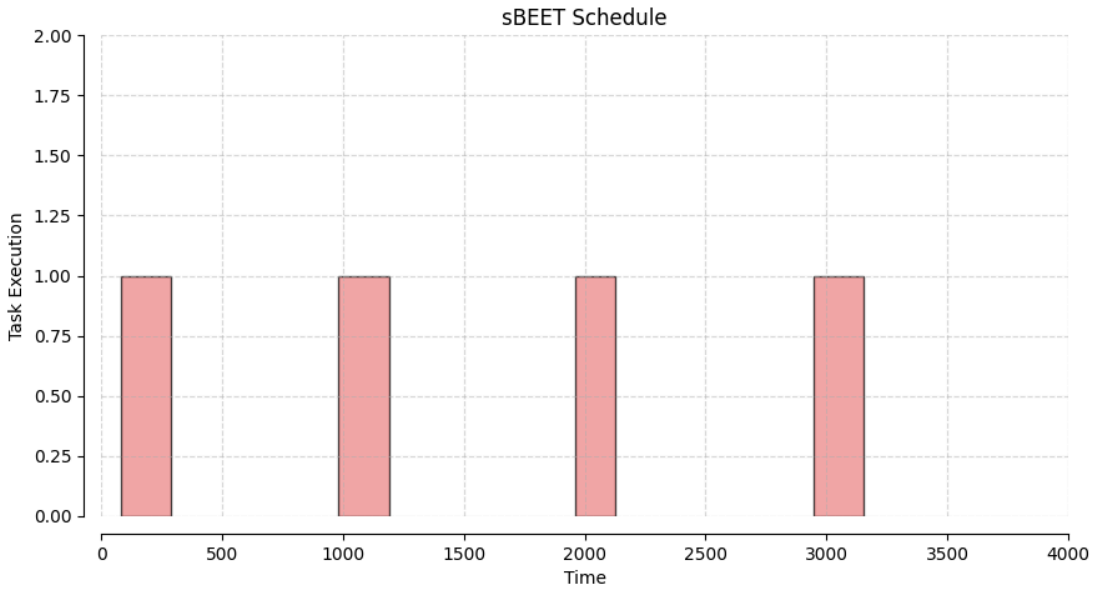
Hist2:


sBEET Schedule
Task Execution
Time
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
500
1000
1500
2000
2500
3000
3500
4000

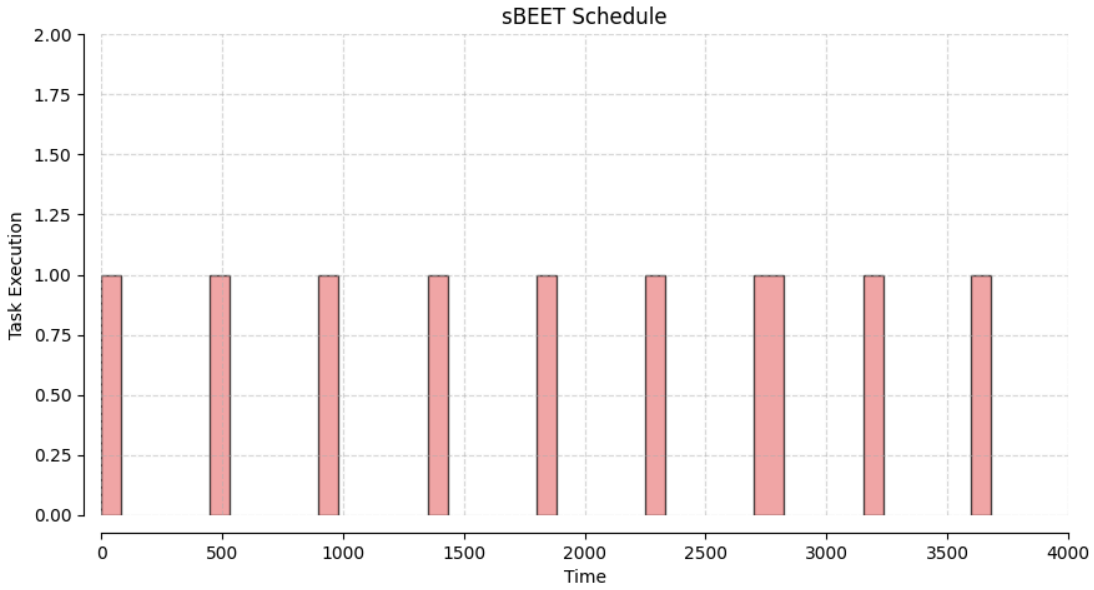
Hotspot:



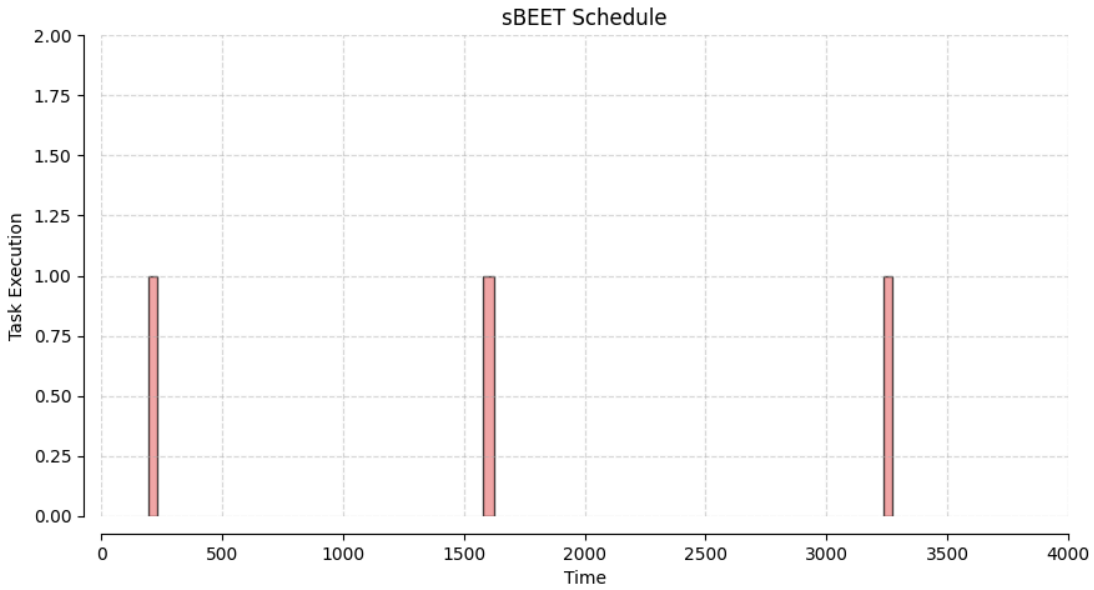
Mmul:



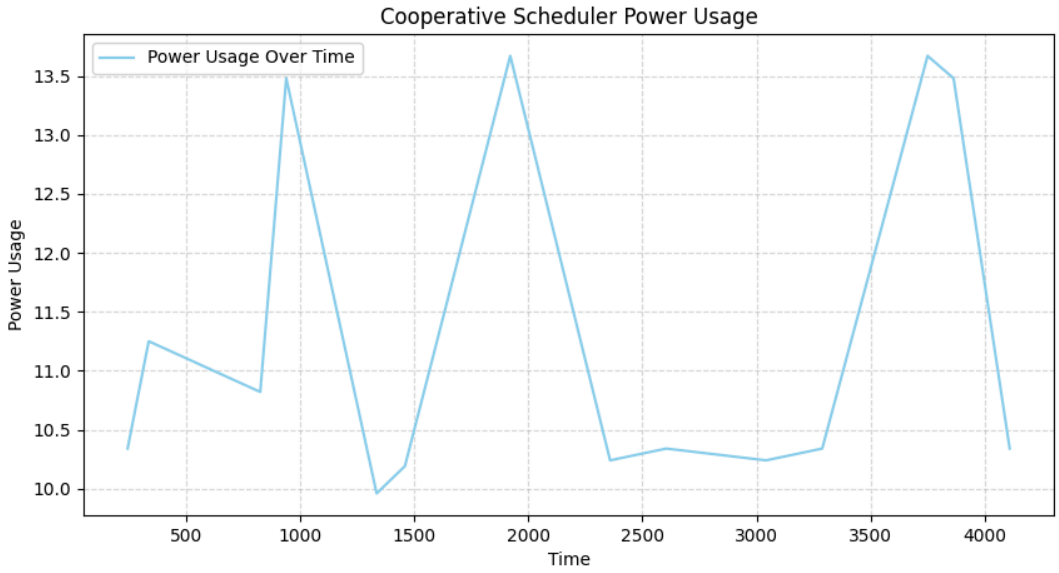
Mmul2:



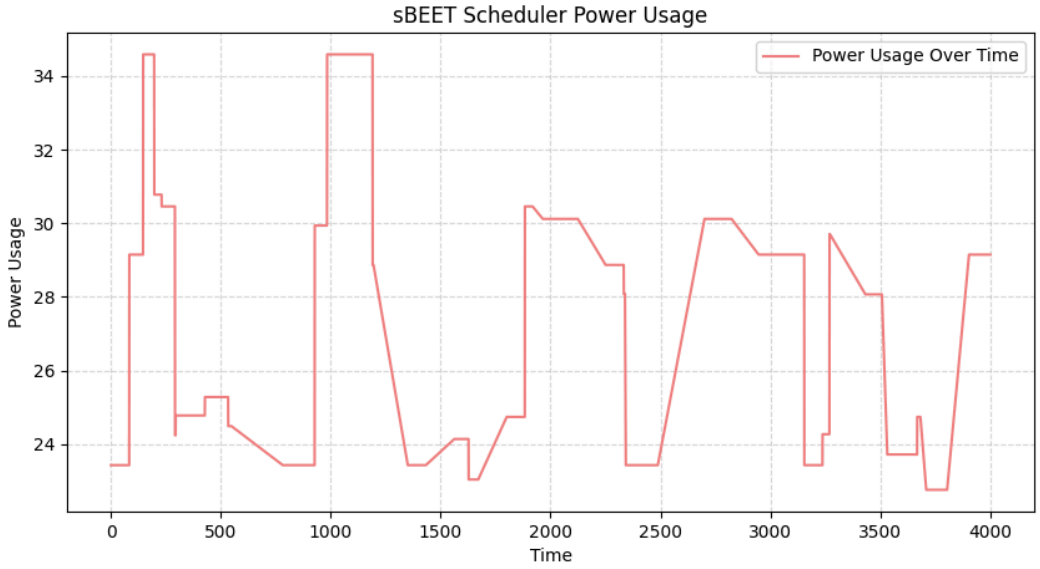
Stereodisparity:



2. نمودار توان مصرفی الگوریتم‌ها

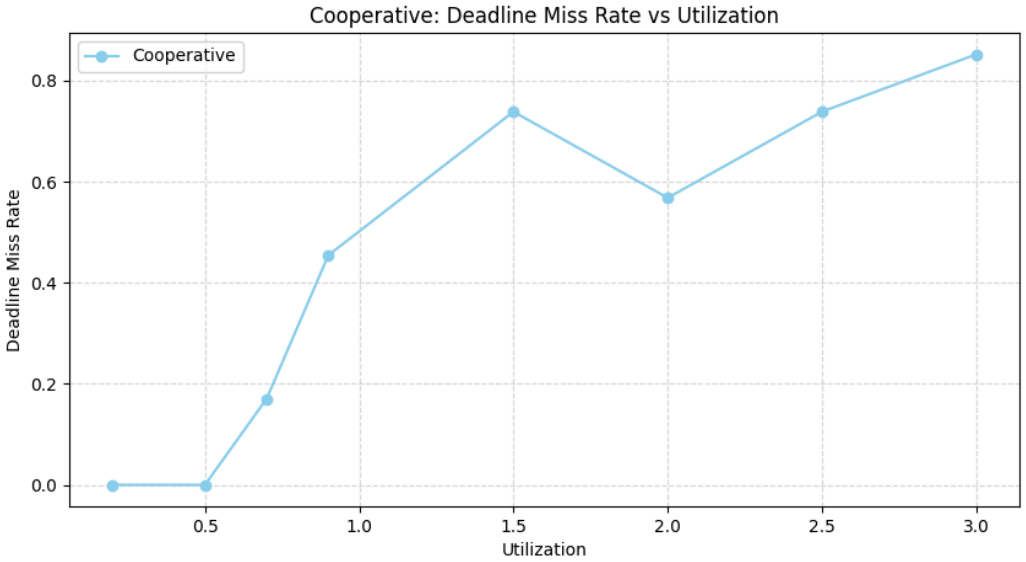


این نمودار مربوط به توان مصرفی الگورتیم Cooperative بر حسب زمان است. همچنین مقدار utilization برابر 2 قرار داده شده است. مصرف توان از حدود 10 واحد شروع می‌شود و به سرعت در حدود 13.5 واحد در نزدیکی 500 واحد زمان به اوج می‌رسد. این اوج اولیه می‌تواند به دلیل هزینه‌های راه‌اندازی یا فرآیندهای اولیه وظایف در زمان‌بند Cooperative باشد. پس از اوج اولیه، مصرف توان به طور قابل توجهی کاهش می‌یابد و در حدود 10.5 تا 11.5 واحد با نوسانات جزئی پایدار می‌شود. پایداری نسبی نشان می‌دهد که زمان‌بند به یک وضعیت متعادل‌تر می‌رسد که مصرف توان در آن ثابت است. یک اوج قابل توجه دیگر در حدود 2500 واحد زمان مشاهده می‌شود که به سطوح مشابه اوج اولیه می‌رسد. اوج‌های دوره‌ای می‌توانند نشان‌دهنده این باشد که در این لحظات تسک‌ها بسیار زیاد شده‌اند. کاهش شدید پس از اوج‌ها نیز نشان می‌دهد که این دوره‌های با تقاضای بالا کوتاه مدت هستند و سیستم به سرعت به حالت توان پایین‌تر باز می‌گردد.



این نمودار مربوط به توان مصرفی الگورتیم sBEET بر حسب زمان است. همچنین در اینجا هم مقدار utilization برابر 2 قرار داده شده است. مصرف توان از حدود 24 واحد شروع می‌شود و به سرعت به حدود 34 واحد می‌رسد. زمان‌بند sBEET مصرف توان اولیه بالاتری نسبت به زمان‌بند Cooperative دارد که احتمالا به دلیل استراتژی اولیه تخصیص و اجرای وظایف آن است. نوسانات قابل توجهی در مصرف توان با اوج‌ها و فرودهای متعدد در طول دوره زمان‌بندی مشاهده می‌شود. نوسانات مکرر نشان می‌دهد که زمان‌بند sBEET به طور مداوم به تقاضاهای وظایف تنظیم می‌شود که منجر به مصرف توان متغیر می‌شود. این مورد در الگوریتم Cooperative هم بود، اما در اینجا، به‌ویژه در زمان‌های اوج مصرف، برای لحظاتی توان به طور ثابت باقی می‌ماند. در اینجا به طور کلی مصرف توان نسبتاً بالا باقی می‌ماند. این مصرف توان بالاتر و تغییرپذیری بیشتر می‌تواند نشان‌دهنده رویکرد زمان‌بندی پویاتری باشد که به تغییرات بار کاری و نیازهای وظایف تطبیق می‌یابد. بنابراین می‌توان گفت الگوریتم Cooperative مصرف توان را پس از اوج اولیه به طور موثرتری متعادل می‌کند که منجر به مصرف توان کلی کمتری می‌شود. پس این الگوریتم برای سیستم‌هایی که کارایی انرژی حیاتی است مناسب باشد. زمان‌بند sBEET با مصرف توان بالاتر و متغیرتر، بهینه‌سازی اجرای وظایف و عملکرد را حتی با هزینه مصرف بالاتر انرژی به عنوان الویت خود قرار می‌دهد. این نوع الگوریتم می‌تواند در سیستم‌هایی که عملکرد اولویت بیش‌تری بر کارایی انرژی دارد، مفید باشد.

3. نمودار مقایسه نرخ Miss Deadline با بهره‌وری

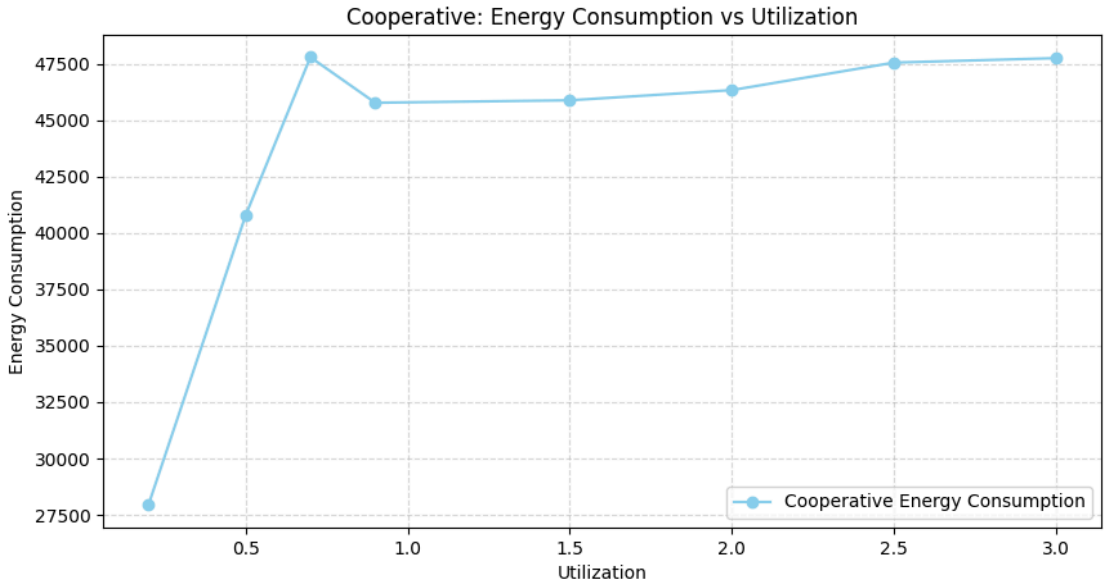


در اینجا نمودار مربوط به Deadline miss rate نسبت به utilization در الگوریتم Cooperative را بررسی کرده‌ایم. در اینجا الگوریتم را برای بهره‌وری از 0.2 تا 3.0 بررسی کرده‌ایم. در مقادیر پایین بهره‌وری (0.2 تا 0.5)، نرخ از دست دادن ددلاین بسیار پایین است که نشان می‌دهد بیشتر وظایف کار خود را در زمان مناسب انجام داده‌اند. با افزایش بهره‌وری به بیش از 0.5، نرخ از دست دادن ددلاین به شدت افزایش می‌یابد.

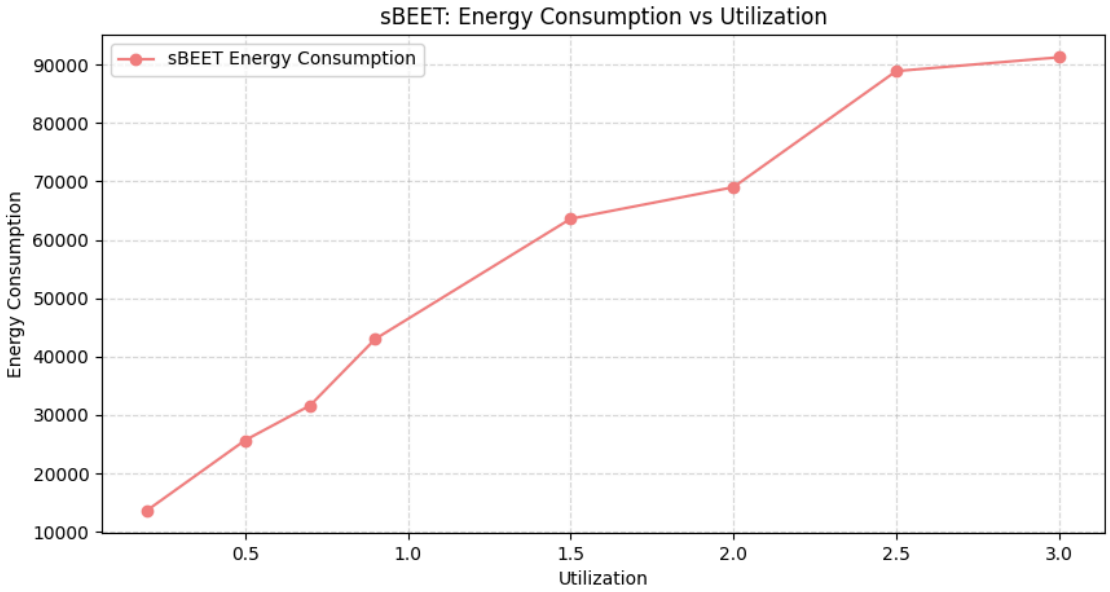
به طور کلی می‌توان گفت زمان‌بند Cooperative در سطوح پایین بهره‌وری عملکرد خوبی دارد اما با افزایش بهره‌وری سیستم دچار مشکل می‌شود. بنابراین زمان‌بند نمی‌تواند بار افزایش یافته را به طور موثر مدیریت کند، که منجر به از دست دادن بیشتر ددلاین‌ها می‌شود. کاهش در بهره‌وری 2.0 می‌تواند به دلیل ویژگی‌های خاص وظایف یا رفتارهای زمان‌بندی باشد که به طور موقت عملکرد را بهبود می‌بخشند.

این نمودار مشابه بخش قبلی است، فقط با این تفاوت که بر روی الگوریتم sBEET پیاده‌سازی شده است. در مقادیر پایین بهره‌وری (0.2 تا 0.9)، نرخ از میس کردن ددلاین بسیار پایین است که نشان‌دهنده عملکرد خوب الگوریتم است. با افزایش بهره‌وری به بیش از 0.9، نرخ از میس کردن ددلاین به تدریج زیاد می‌شود و به مرور به سرعت آن هم افزوده می‌شود. اما باید توجه کنیم که این افزایش به طور تدریجی‌تری نسبت به زمان‌بند Cooperative است. این نشان می‌دهد که sBEET تحت بارهای بالاتر مقاومت بیشتری دارد و در رسیدن به ددلاین‌ها عملکرد بهتری دارد. زمان‌بندی Cooperative چون شامل رویکردهای ساده‌تر یا استاتیک‌تر برای تخصیص وظایف باشد نتایج ناکارآمدی‌ها در بارهای بالا دارد. از سوی دیگر، sBEET به دلیل اینکهتکنیک‌های زمان‌بندی دینامیک‌تر و تطبیقی‌تری را شامل می‌شود اجازه می‌دهد تا بار را بهتر متعادل کند و وظایف را اولویت‌بندی کند، که منجر به عملکرد بهتر این الگوریتم می‌شود.

4. نمودار مقایسه انرژی مصرفی با بیشترین بهره‌وری

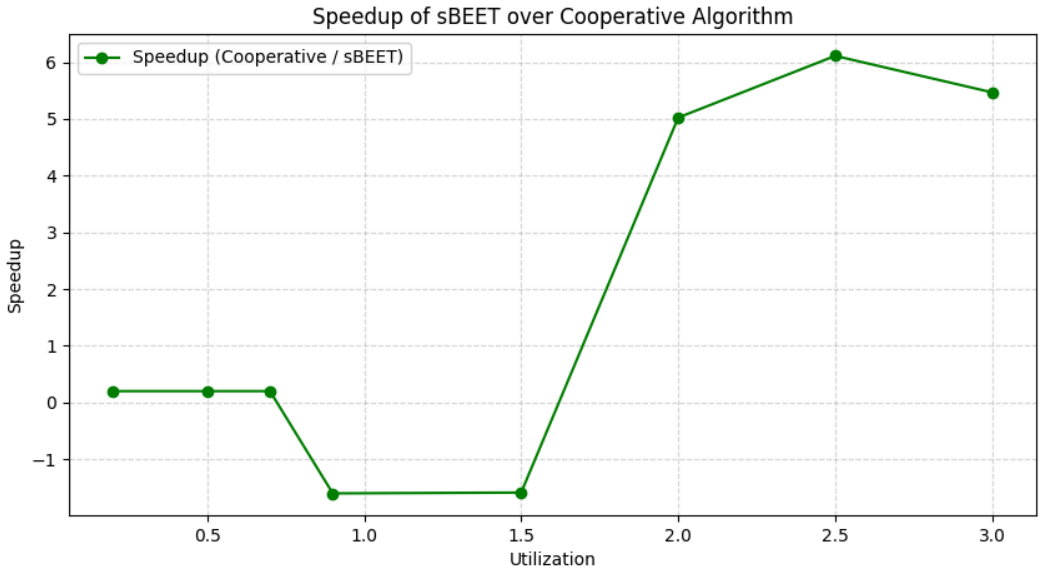


این نمودار مربوط به میزان انرژی مصرفی نسبت به بهره‌وری در الگوریتم Cooperative است. در بهره‌وری پایین (0.2)، مصرف انرژی نسبتاً کم است. با افزایش بهره‌وری از 0.2 به 0.5، مصرف انرژی به شدت افزایش می‌یابد. افزایش شدید مصرف انرژی از 0.2 به 0.5 نشان می‌دهد که زمان‌بند تعاونی برای مدیریت تعداد بیشتری از وظایف به انرژی بیشتری نیاز دارد. بین سطوح بهره‌وری 0.5 تا 1.5، مصرف انرژی نسبتاً پایدار باقی می‌ماند و در بهره‌وری 1.0 کمی کاهش می‌یابد. پایداری در مصرف انرژی در حدود بهره‌وری 1.0 نشان‌دهنده این است که زمان‌بند ممکن است به یک محدوده بهره‌وری کارآمدتر وارد شده باشد که وظایف اضافی به طور متناسب مصرف انرژی را افزایش نمی‌دهند. از 1.5 تا 3.0، افزایش تدریجی در مصرف انرژی مشاهده می‌شود، اما نرخ افزایش کمتر از افزایش اولیه از 0.2 به 0.5 است. این می‌تواند به این دلیل باشد که با افزایش بهره‌وری، سیستم ممکن است بهتر از قبل بهینه‌سازی انرژی را انجام دهد.



این نمودار مربوط به میزان انرژی مصرفی نسبت به بهره‌وری در الگوریتم sBEET است. در بهره‌وری پایین (0.2)، مصرف انرژی بسیار کم است. با افزایش بهره‌وری، افزایش پیوسته و یکنواختی در مصرف انرژی مشاهده می‌شود. افزایش پیوسته و یکنواخت در مصرف انرژی نشان می‌دهد که زمان‌بند sBEET مصرف انرژی را به طور پیش‌بینی‌پذیر و منظم با افزایش بار سیستم مقیاس می‌دهد. این افزایش تقریباً خطی تا بالاترین سطح بهره‌وری (3.0) ادامه دارد. اما باید به این نکته هم توجه داشته باشیم که میزان مصرف انرژی به طور کلی در این الگوریتم بیش‌تر است و به مقدار قابل توجهی هم بیش‌تر است. به طور کلی زمان‌بندی Cooperative در سطوح پایین بهره‌وری دارای سربار بیشتری باشد که منجر به مصرف انرژی بیشتر اولیه می‌شود. روند خطی sBEET نشان می‌دهد که از استراتژی کارآمدتری برای مدیریت انرژی با افزایش بار سیستم استفاده می‌کند که منجر به عملکرد بهتر در مصرف انرژی می‌شود. انتخاب sBEET برای سیستم‌هایی که استفاده پیش‌بینی‌پذیر و کارآمد از انرژی حیاتی است، به ویژه با افزایش بار، مزیت بیشتری دارد. از طرف دیگر، زمان‌بند Cooperative برای شرایطی که انتظار می‌رود بار سیستم در حدود سطوح متوسط تثبیت شود، که در آن مصرف انرژی آن پایدار می‌شود، مناسب باشد. از طرفی برای زمان‌هایی که مصرف انرژی برای ما اهمیت بیش‌تری دارد، Cooperative گزینه مناسب‌تری است.

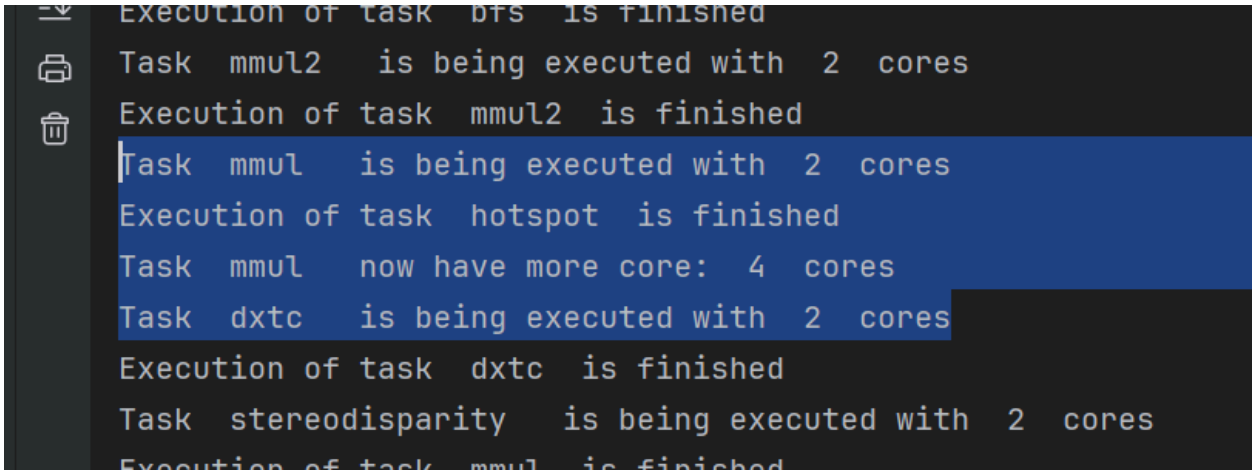
5. نمودار سرعت (Speedup) نسبت به الگوریتم Cooperative با افزایش بهره‌وری



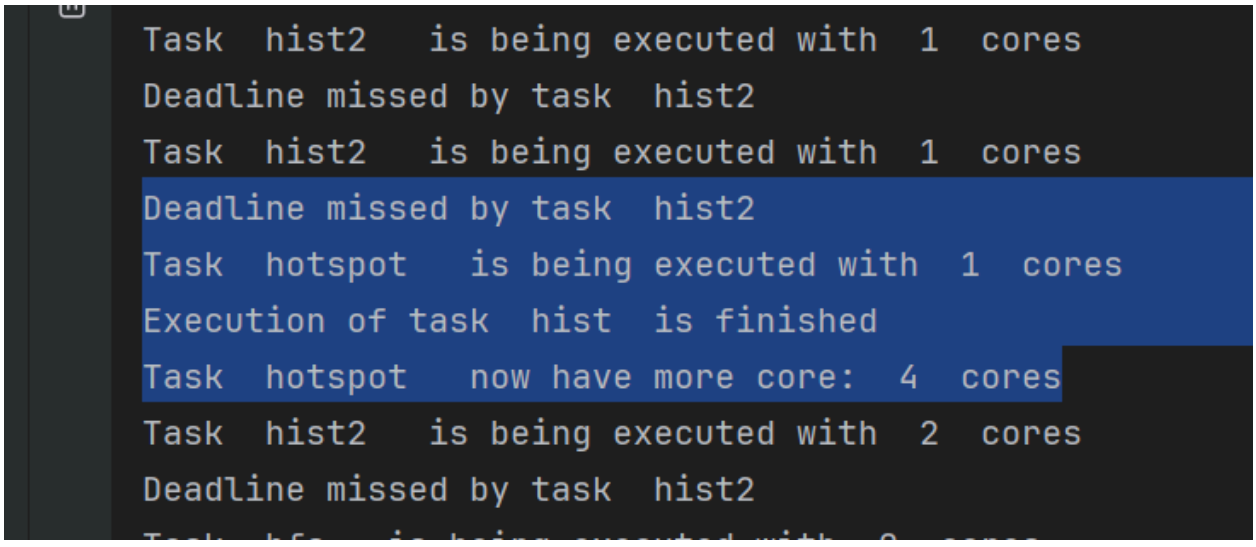
این نمودار مربوط به speed up بر حسب بهره‌وری‌های مختلف است. برای محاسبه اسپیدآپ نسبت زمان اجرای تسک‌ها در حالت Cooperative به sBEET را به دست آورده‌ایم. در مقادیر پایین بهره‌وری (0.2 تا 0.7)، اسپیدآپ حدود 0 است که نشان‌دهنده تفاوت عملکرد ناچیز بین دو زمان‌بند است. در حدود بهره‌وری 1.0، اسپیدآپ به زیر صفر می‌رود که نشان می‌دهد زمان‌بند Cooperative در این محدوده بهتر عمل می‌کند. البته ممکن است به طور خاص در این تسک ست خاص همچین نتیجه‌ای را داشته باشیم. افزایش شدید اسپیدآپ از 1.5 به بعد نشان می‌دهد که sBEET با افزایش بار سیستم به طور قابل توجهی کارآمدتر می‌شود. به طور کلی می‌توان گفت sBEET در سطوح بالاتر بهره‌وری برتری خیلی بیش‌تری دارد و آن را به انتخاب بهتری برای سیستم‌هایی که انتظار می‌رود تحت بارهای سنگین عمل کنند تبدیل می‌کند. زمان‌بند Cooperative، اگرچه عموماً کمتر کارآمد است، ممکن است در شرایط خاصی که در آن عملکرد بهتری نسبت به sBEET دارد، به ویژه در سطوح متوسط بهره‌وری، مؤثرتر باشد.

6. حالت خاص در الگوریتم

با توجه به این که ما الگوریتم را طوری تغیر دادیم که در صورتی که یکی از دو تسک در حال اجرا اتمام یابند برای ادامه ی کار الگوریتم دو تسک موجود برای اجرا (تسک در حال اجرای قبلی و تسک انتخاب شده ی جدید) را به لحاظ execution time و power برسی میکند و به بهینه ترین حالت ممکن کور هارا تخصیص میدهد. برای مثال تکه هایی از خروجی پروژه را انتخاب میکنیم:



در این قسمت با اتمام کار تسک hotspot تعداد کور های تسک mmul دو واحد افزایش میابد و تسک جدید با 2 کور شروع به کار میکند.



در این بخش با اتمام کار تسک hist تعدا کور های تسک hotspot سه واحد افزارش پیدا کرده و تسک بعدی با 2 کور شروع به کار میکند.

نتیجه

انتخاب بین الگوریتم‌های sBEET و Cooperative بستگی به نیازهای خاص سیستم دارد. اگر هدف اصلی صرفه‌جویی در مصرف انرژی باشد، الگوریتم Cooperative گزینه بهتری است. این الگوریتم به ویژه در بهره‌وری‌های پایین‌تر مصرف انرژی کمتری دارد. از سوی دیگر، اگر سرعت اجرا در بهره‌وری‌های بالا مهم باشد، sBEET می‌تواند مزایای بیشتری ارائه دهد. با این حال، باید به افزایش مصرف انرژی و نرخ از دست رفتن ددلاین در بهره‌وری‌های بالا نیز توجه داشت.

در نهایت، برای بهینه‌سازی سیستم زمانبندی، باید اولویت‌های خاصی مانند انرژی مصرفی، سرعت اجرا و نرخ از دست رفتن ددلاین در نظر گرفته شوند و بر اساس این اولویت‌ها، الگوریتم مناسب انتخاب شود. ترکیبی از این الگوریتم‌ها نیز ممکن است در شرایط مختلف به کار گرفته شود تا بهترین نتایج حاصل شود.