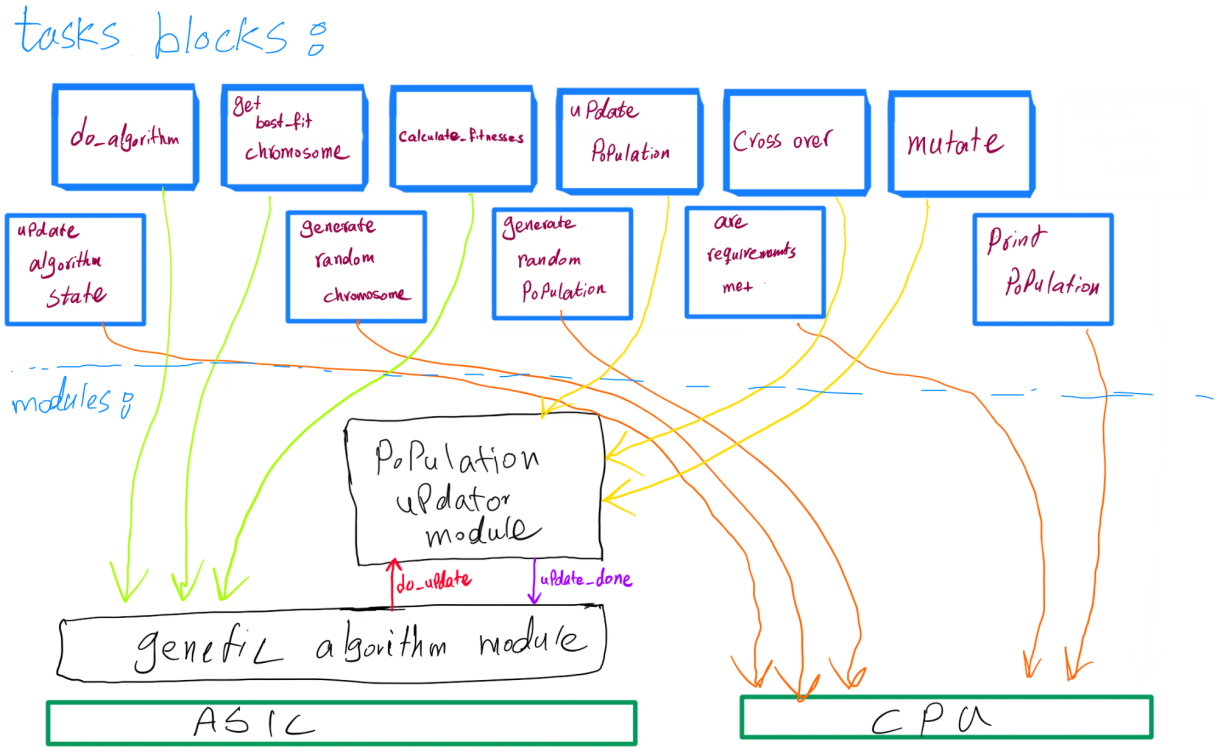
همطراحی سخت افزار و نرم افزار

سینا رستمی 9822143

در این پروژه الگوریتم ژنتیک را به صورت طراحی توام سخت‌افزار و نرم‌افزار پیاده کرده‌ایم.
برای این کار ابتدا خود الگوریتم ژنتیک را پیاده سازی کرده و قسمت های مختلف آن را مورد بررسی قرار می‌دهیم تا کارهای سبک و کم تکرار را به بخش نرم افزار و کارهای سنگین و پرتکرار را به بخش سخت افزار نسبت دهیم.
با مشخص کردن بلاک های مسئله و بررسی آنها به تقسیم بندی زیر به ۳ ماژول می‌رسیم:



در ادامه به بررسی دلایل تخصیص هر تسک می‌پردازیم.

Tasks

- Do_algorithm:

این تسک یکبار اجرا می‌شود ولی به دلیل وابستگی داده ای زیادی که به تسک های دیگر دارد، در ماژول genetic_algorithm قرارش می‌دهیم تا سریع تر باشد.

- Get bestfit chromosome

این تسک به تعداد زیاد اجرا می‌شود و نیازمند زیرساخت سریع برای اجرا شدنش هستیم. به طوری که در هر مرحله از الگوریتم ژنتیک یکبار همه کروموزوم ها را بررسی می کند و کروموزومی که بیشترین مقدار fitness را دارد بر می گرداند. پس این تسک را نیز روی ماژول genetic_algorithm قرار می‌دهیم.

- Calculate fitnesses

این تسک نیز به تعداد زیاد اجرا می‌شود و نیازمند زیرساخت سریع برای اجرا شدنش هستیم. به طوری که در هر مرحله از الگوریتم ژنتیک یکبار همه کروموزوم ها را بررسی می کند و با توجه به حضور یا عدم حضور هر ژن، مقدار fitness را برای هر کروموزوم حساب می کند. پس این تسک را نیز روی ماژول genetic_algorithm قرار می‌دهیم.

- Update population

این تسک بعد از هر مرحله از الگوریتم ژنتیک اجرا می‌شود و وظیفه آن این است که population را آپدیت کند. به این شکل عمل می کند که ابتدا کروموزم ها را به ترتیب fitnessشان مرتب کرده و سپس دوتا دوتا آنها را به روشی که در مستند پروژه گفته شده آپدیت می کند. این تسک نیز بار پردازشی بالایی دارد پس باید روی ماژول سخت افزاری قرار گیرد. اما از آنجایی که شیوه کار آن مستقل از ماژول سخت افزاری قبلی بود، یک ماژول سخت افزاری جدید در نظر می گیریم که وظیفه آن اختصاصا آپدیت کردن population باشد. نام این ماژول population updator module است. این ماژول از طریق event با ماژول اصلی الگوریتم ارتباط برقرار می کند.

- Crossover
- Mutate

از آنجایی که در فرآیند آپدیت کردن population این دو تسک به دفعات زیادی اجرا می‌شوند و داری ارتباط و وابستگی داده ای بسیاری با تسک قبلی هستند. این دو تسک را نیز در ماژول population updator module کنار تسک قبلی قرار می‌دهیم تا به سرعت با هم ارتباط برقرار کرده و اجرا شوند.

- Update algorithm state

این تسک بعد از هر بار آپدیت کردن population فقط یکبار انجام می‌شود. پس دارای تراکم اجرای پایین است. و دارای اندک وابستگی داده ای با ماژول genetic algorithm است. اما از آنجایی که تعداد دفعات اجرایش کم است از این وابستگی چشم پوشی کرده و این تسک را به ماژول نرم افزاری می‌بریم.

- Generate random population
- Generate random chromosome

این دو تسک در ابتدای شروع الگوریتم یکبار اجرا می‌شوند و وظیفه آنها ایجاد population رندوم اولیه برای شروع الگوریتم ژنتیک است. از آنجایی که فقط یکبار اجرا می‌شوند و بار پردازشی چندان بالایی ندارند. این دو تسک را روی ماژول نرم افزار قرار می‌دهیم.

- Are requirements met

این تسک نیز همانند تسک update algorithm state در هر بار آپدیت کردن population فقط یکبار انجام می‌شود و پردازش سبکی دارد. وظیفه آن این است که بررسی کند که آیا ما به شرط پایان رسیده ایم یا خیر؟! پس این تسک را نیز روی ماژول نرم افزار می گذاریم.

- Print population

برای اینکه از نحوه اجرا الگوریتم آگاه باشیم، نیاز است که پس از هر بار آپدیت کردن population وضعیت این نمونه را ببینیم. پس تسکی تعریف کردیم که یک population را دریافت کرده و وضعیت آن را پرینت می کند. اما از آنجایی که توان پردازشی بالایی ندارد و همچنین تعداد دفعات اجرا شدن آن کم است، این تسک را نیز روی ماژول نرم‌افزار قرار می‌دهیم.

Report

برای بررسی صحت عملکرد سیستم طراحی شده، به حل مسئله knapsack توسط این سیستم پرداختیم.
این مسئله به این شرح است که تعدادی شئ داریم که میخواهیم آنها را درون یک کوله پشتی بگذاریم که هر شئ یک وزن و یک ارزش دارد و کوله پشتی هم حداکثر وزن قابل تحمل دارد. این مسئله پاسخ به این سوال است که کدام اشیاء را برداریم و در کوله پشتی بگذاریم که مجموع وزن این اشیاء از حداکثر وزن قابل تحمل کوله پشتی بیشتر نشود و همچنین مجموع ارزش اشیاء درون کوله پشتی بیشترین مقدار ممکن باشد.

به این شکل ورودی های مسئله را تعریف کردیم:

```
std::vector<int> values({1, 2, 1, 3, 1});
std::vector<int> weights({2, 1, 4, 5, 5});
// maximum fitness : 0.5 + 2 + 0.6 = 3.1
```

```
// knap-sack extra condition
weight_limit.write(8);
```

حال با شرایط مختلف پایان، این نمونه را اجرا کردیم و خروجی های متناظر به شکل زیر حاصل شدند:

۱. با شرط حداقل fitness برابر 3.0

```
// finish condition
finish_condition.finish_condition_type = FinishConditionType::REQUIRED_FITNESS;
finish_condition.required_fitness = 3.0;
```

خروجی به همراه log های هر مرحله از الگوریتم:

```
sina@sina ~/Desktop/uni/4001/codesign-hw-sw/final_project/src/build <master●>
$ ./final_project

        SystemC 2.3.3-Accellera --- Oct 24 2021 15:23:56
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED

Warning: (W506) illegal characters: genetic algo substituted by genetic_algo
In file: ../../../src/sysc/kernel/sc_object.cpp:247
[0, 1, 0, 1, 1, ] : 0
[0, 0, 0, 1, 0, ] : 0.6
[1, 1, 0, 0, 1, ] : 2.7
[0, 0, 1, 1, 1, ] : 0


[1, 1, 0, 0, 1, ] : 2.7
[0, 0, 0, 1, 0, ] : 0.6
[0, 1, 0, 1, 1, ] : 0
[0, 0, 1, 1, 1, ] : 0


[0, 0, 0, 0, 1, ] : 0.2
[1, 1, 0, 1, 0, ] : 3.1
[0, 1, 0, 1, 1, ] : 0
[0, 0, 1, 1, 1, ] : 0


answer : [1, 1, 0, 1, 0, ] : 3.1
```

۲. با شرط اجرای الگوریتم به تعداد ۴ مرحله

```
// finish condition
finish_condition.finish_condition_type = FinishConditionType::NUMBER_OF_UPDATES;
finish_condition.number_of_updates = 4;
```

خروجی به همراه log های هر مرحله از الگوریتم:

```
sina@sina ~/Desktop/uni/4001/codesign-hw-sw/final_project/src/build <master●>
$ ./final_project

        SystemC 2.3.3-Accellera --- Oct 24 2021 15:23:56
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED

Warning: (W506) illegal characters: genetic algo substituted by genetic_algo
In file: ../../../src/sysc/kernel/sc_object.cpp:247
[1, 1, 0, 0, 0, ] : 2.5
[1, 0, 1, 1, 0, ] : 0
[0, 0, 0, 1, 1, ] : 0
[1, 0, 1, 0, 1, ] : 0


[1, 1, 0, 0, 0, ] : 2.5
[1, 0, 1, 1, 0, ] : 0
[0, 0, 0, 1, 0, ] : 0.6
[1, 0, 1, 0, 1, ] : 0


[0, 0, 0, 1, 0, ] : 0.6
[1, 1, 0, 0, 0, ] : 2.5
[1, 0, 1, 1, 0, ] : 0
[1, 0, 1, 0, 1, ] : 0


[0, 1, 0, 0, 0, ] : 2
[1, 0, 0, 1, 0, ] : 1.1
[1, 0, 1, 1, 0, ] : 0
[1, 0, 1, 0, 1, ] : 0


answer : [0, 1, 0, 0, 0, ] : 2
```

۳. با شرط همگرایی جواب پس از ۳ مرحله پیاپی

```
// finish condition
finish_condition.finish_condition_type = FinishConditionType::CONVERGENCY_OF_FITNESS;
finish_condition.convergence_accuracy = 3;
```

خروجی به همراه log های هر مرحله از الگوریتم:

```
sina@sina ~/Desktop/uni/4001/codesign-hw-sw/final_project/src/build <master●>
$ ./final_project

        SystemC 2.3.3-Accellera --- Oct 24 2021 15:23:56
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED

Warning: (W506) illegal characters: genetic algo substituted by genetic_algo
In file: ../../../src/sysc/kernel/sc_object.cpp:247
[1, 0, 0, 0, 0, ] : 0.5
[0, 1, 1, 1, 0, ] : 0
[0, 0, 1, 1, 0, ] : 0
[1, 0, 0, 0, 1, ] : 0.7


[1, 0, 0, 0, 1, ] : 0.7
[1, 0, 0, 0, 0, ] : 0.5
[0, 0, 1, 1, 0, ] : 0
[0, 1, 1, 1, 0, ] : 0


[1, 0, 0, 0, 1, ] : 0.7
[1, 0, 0, 0, 0, ] : 0.5
[0, 0, 1, 1, 0, ] : 0
[0, 1, 1, 1, 0, ] : 0


answer : [1, 0, 0, 0, 1, ] : 0.7
```