دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

گزارش تحقیق درس VLSI پیشرفته

بررسی و مقایسه روش های مبتنی بر یادگیری عمیق در مسئله Placement و Routing برای مدارهای VLSI

نگارش

رضا آدینه پور

استاد درس

جناب آقای دکتر صدیقی

بهمن ۱۴۰۲

چکیده

جایگذاری قطعات و سیم‌کشی مدارهای مجتمع در مقیاس های خیلی بزرگ، همواره یکی از چالشی ترین مراحله ها در فرایند طراحی مدار است. در گذشته این دو مرحله به صورت دستی توسط اپراتور انسانی انجام می‌شد. به طوری که ممکن بود بار و بارها طراحی انجام شده به دلیل برخی از ملاحظات فنی عوض می‌شد و این تغییر دادن طراح و سیم‌کشی مجدد آن تایم زیادی را نیاز دارد که با بزرگ شدن طراحی‌ها در مدارهای مجتمع امروزی فرایندی سخت و حتی شاید غیر ممکن به نظر برسد. پیشرفت تکنولوژی در حوزه هوش مصنوعی و یادگیری عمیق، دنیای طراحی مدارهای مجتمع را هم دستخوش تغییراتی کرده است. به‌طوری که امروزه دنیای طراحی آیسی به سمتی می‌رود که فرایند جایگذاری و سیم‌کشی آیسی بدون دخالت انسال و به‌طور کاملا خودکار با دقت بالا و خطای بسیار کم انجام شود.

در این گزارش به بررسی، بیان مزایا و معایب دو روش مشابه برای این‌کار که در مقاله های [۱] و [۲] پیشنهاد شده است پرداخته ایم.

کلیدواژه‌ها: جایگذاری، سیم‌کشی، یادگیری عمیق

فهرست مطالب

فهرست جداول

فهرست تصاویر

فصل ۱

مقدمه

مسئله چینش[۱] و سیم‌کشی[۲] از گذشته تا به امروز جزئی از مهمترین مسائل طراحی آیسی است. با پیشرفت شبکه‌های عصبی مصنوعی و استفاده گسترده آنها در کاربردها و اپلیکیشن‌های متفاوت محققان حوزه طراحی IC در صدد برآمدند که بخش چینش و سیم‌کشی طراحی را که یکی از وقت‌گیر ترین مراحل طراحی است را به کمک شبکه‌های عصبی انجام دهند.

۱-۱ تعریف مسئله

مراحل طراحی IC را می‌توان به ۴ مرحله زیر تقسیم کرد:

۱. طراحی شماتیک بخش‌های مختلف مدار

۲. آنالیز و بررسی طراحی انجام شده و اطمینان از صحت عملکرد مدار به وسیله نرم‌افزار های شبیه‌سازی مانند SPICE و Cadence

۳. مرحله Layout که شامل Placement و Routing است

۴. ساخت آیسی یا Fabrication

مرحله ۱ و ۲ باید به صورت دستی و توسط انسان انجام شود. چرا که شخص طراح می‌بایست بر همه بخش‌های طراحی خود مسلط باشد و بتواند اگر نیاز بود بخش‌های دیگیری به طراحی اضافه و یا از آن کم

[۱] Placement
[۲] Routing

شود، آن را اعمال کند. اما هوش مصنوعی به این مرحله نیز وارد شده است و ابزارهایی مانند EDA Magic[۳] مخصوص این کار آموزش داده شده است که با دادن اطلاعات مورد نیاز خود برای طراحی، مدار مورد نیاز ما را به صورت کامل طراحی می‌کند. که در این گزارش به آن نمی‌پردازیم.

در مرحله ساخت آیسی[۴] نیز هوش‌مصنوعی به صورت محدود وارد شده است و همچنان مرحله ساخت به صورت قدیمی و سنتی انجام می‌شود.

در گذشته، در مرحله ۳، طراحی ها با استفاده از ابزارهای کامپیوتری CAD[۵] انجام می‌شود. از مزایا ابزارهای CAD می‌توان به موارد زیر اشاره کرد:

- تحلیل دقیق
- تولید خروجی باکیفیت
- پشتیبانی گسترده نرم‌افزاری

اما در کنار مزایای نامبرده می‌توان به معایب آن هم اشاره کرد:

- لزوم وجود کاربر انسانی[۶] برای انجام طراحی
- زمان زیاد برای انجام
- هزینه بسیار بالای ابزارهای CAD

با پیشرفت ابزارهای هوش‌مصنوعی مانند شبیه‌های عصبی[۷] ابزارهای مختلفی که برپایه شبکه‌های عصبی کار می‌کنند معرفی شده است. این ابزارها با حذف اپراتور انسانی در فرایند Place&Route و کاهش زمان انجام این فاز از طراحی، کمک بزرگی به این زمینه کرده است.

۱-۲ اهمیت موضوع

در طراحی‌های تجاری، طراح‌ها مجبوراند چندین بار طراحی خود را برای دست‌یابی به بهترین و بهینه[۸] ترین حالت عوض کنند. استفاده از روش‌های طراحی سنتی قدیمی، برای مدارهای بزرگ[۹] امروزی، بسیار فرایندی

[۳] برای اطلاعات بیشتر می‌توان به اینجا مراجعه کرد: snapmagic.com
[۴] Fabrication
[۵] Computer Aided Design
[۶] Designer
[۷] Neural Network
[۸] Optimum
[۹] Complex

طولانی و کند است که فرایند تکرار طراحی برای دستیابی به بهینه‌ترین حالت جایگذاری و سیم‌کشی را به شدت کند می‌کند.

۱-۳ اهداف پژوهش

در اصل، در این مقالات، یک مسئله بهینه‌سازی غیر خطی حل شده است و به مسئله جایگذاری و سیم‌کشی به عنوان یک فرایند غیرخطی نگاه شده است که قرار است آن را بهینه کنیمو به طوری اهداف ما یعنی پیدا کردن بهترین محل قرارگیری سلول‌های طراحی با حداقل سیم‌کشی ممکن که کمترین همپوشانی را داشته باشد ارضا شود.

فصل ۲

مقاله [۱]

۲-۱ ایده اصلی مقاله

در این مقاله، به مسئله Placement با رویکرد یک مسئله شبکه عصبی نگاه شده است و تلاش شده است که بهینه ترین حالت ممکن پیدا شود. خروجی این مقاله منجر به ارائه یک چارچوب[۱] متن باز[۲] مبتنی بر CPU و GPU با استفاده از شبکه‌های عمیق و زبان برنامه نویسی Python و C++ و کتابخانه های PyTorch و CUDA شده است که بدون افت کیفیت نسبیت به روش‌های قدیمی، تا ۳۰ برابر افزایش سرعت برای مسئله چینش ارائه شده است.

۲-۲ کارهای پیشین

روش ها و الگوریتم هایی که تا اکنون توسعه داده شده است، به نام پیاده‌سازی تحلیلی[۳] معروف است. این روش ها خروجی‌های با کیفیتی تولید می‌کنند اما مشکل عمده این روش ها، کند بودن آنهاست. معمولا برای افزایش سرعت در این روش ها از تکنیک‌های چندنخی[۴] کردن CPU استفاده می‌شود. روش‌های مبتنی بر چندنخی، سرعت جایگذاری را تا ۵ برابر افزایش می‌دهند اما مشکل عمده آنها این است که کیفیت طرح خروجی، بین ۲ الی ۶ درصد کاهش می‌یابد. [۳، ۴، ۵]

یعنی الگوریتم‌های توسعه داده شده مبتنی بر چند نخی، صرفا از جهت افزایش سرعت بهینه کار می‌کنند و

[۱] Framework
[۲] Open Source
[۳] Analytical placement
[۴] Multi-thread

توجه اصلی بر روی افزایش سرعت است. اما توجهی کمتری نسبت به بهبود کیفیت خروجی دارند و همین امر موجب نا کارامد بودن چنین الگوریتم‌هایی می‌شود.

دسته دیگری از الگوریتم‌ها بر بستر GPU توسعه داده شده است.[۶] این الگوریتم بر اساس خوشه‌بندی[۵] انجام شده است که با موازی‌سازی بر بستر GPU، به‌طور میانگین، سرعت تا ۱۵ برابر نسبت به روش Ana-lytical افزایش یافته است. درصد افت کیفیت نیز برای این الگوریتم، کمتر از ۱ درصد گزارش شده است. اما این روش نیز به دلیل هزینه بالا برای فراهم کردن GPU مورد اقبال واقع نشده است. [۷]

در این مقاله روشی که توسعه داده شده است، بر مبنای روش‌های تحلیلی نام برده شده است. با این تفاوت که این الگوریتم هم در بستر GPU و هم در بستر CPU شتابدهی شده است. که از این بابت عام منظوره بودن الگوریتم و هزینه پایین آن نسبت به سایر الگوریتم های موجود را نشان می‌دهد.

این الگوریتم که به نام DREAM-Place نام‌گذاری شده است، به صورت عمومی[۶] توسعه داده شده است که با سایر الگوریتم‌های جای‌گذاری تحلیلی مثل NTUplace سازگار است. [۸]

ایده‌های اصلی[۷] مقاله به صورت زیر عنوان شده است:

- ایجاد دیدگاهی کاملا جدید برای ارتباط دنیای طراحی آیسی با دنیای هوش‌مصنوعی و یادگیری عمیق به صورت کاملا متن باز برای توسعه در CPU و GPU
- محاسبه بهترین محل قرارگیری سلول ها با کمترین طول و چگالی سیم.
- بهبود سرعت چینش بدون افت کیفیت خروجی تا ۳۰ برابر. به طوری که طراحی ای با ۱ میلیون سلول در بستر CPU در یک دقیقه تمام می‌شود. که این تایمینگ به صورت خطی با افزایش تعداد سلول ها تا حداکثر ۱۰ میلیون تغییر می‌کند.

تمام سورس‌کد نوشته شده برای این الگ.ریتم نیز در گیت‌هاب[۸] قابل دریافت است.

۳-۲ مراحل انجام الگوریتم

قبل از بررسی مراحل انجام، نیاز است که برخی از اصطلاحات این حوزه را بیان کنیم.

جاگذاری تحلیلی[۹] دارای ۳ مرحله اصلی است:

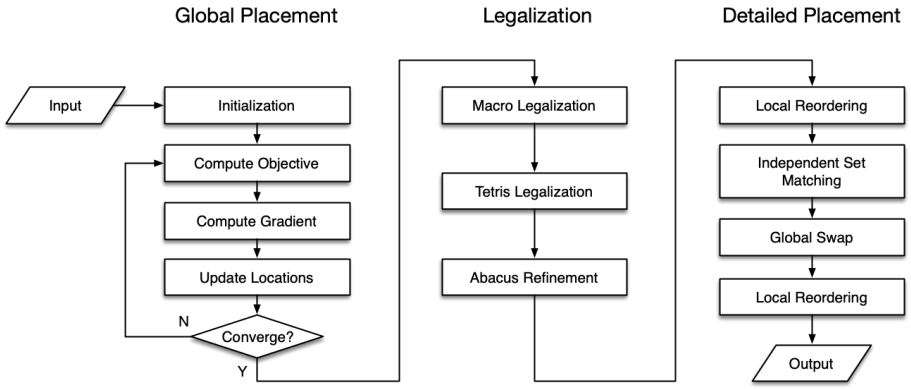
---

[۵] Clustering
[۶] Generic
[۷] Contributions
[۸] github.com/limbo018/DREAMPlace
[۹] Analytical placement

- چینش کلی یا GP[۱۰]: قرار دادن سلول‌ها در طرح با هدف بهینه شدن GP و مینیمم Routing
- بررسی قوانین یا LG[۱۱]: بررسی طراحی انجام شده در مرحله GP و حذف همپوشانی ها و تراز کردن مشکلات طرح.
- چینش جزئی یا DP[۱۲]: بررسی دقیق تر طراحی انجام شده برای رسیدن به دقت و کیفیت بالا در طراحی.

معمولا مرحله GP بیشترین زمان را در فرایند صرف می‌کند به همین دلیل، تمام الگوریتم‌های بیان شده در قبل و همچنین این الگوریتم، نتایج مورد بحث، در مورد مینیمم کردن طول مسیر ها و چگالی آنهاست.

مراحل انجام این الگوریتم را می‌توان به صورت زیر بیان کرد:



شکل ۲-۱: ساختار الگوریتم

در ایتدا و در مهمترین فاز، ورودی‌ها که شامل محل قرارگیری سلول هاست، به شبکه داده می‌شود و شبکه که در ابتدا به یک سری وزن های رندوم مقدار دهی شده است، آموزش می‌بیند و خروجی آن که مقدار خطای محاسبه شده است، با استفاده از الگوریتم پس انتشار خطا[۱۳] به لایه های عقب تر انتشار داده می‌شود تا اینکه بهینه ترین خطا (مینیمم ترین حالا خطا) را پیدا کنیم. «شکل ۲-۲»

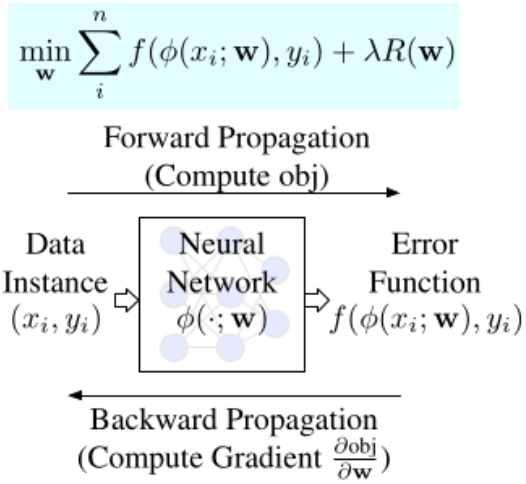
مراحل انجام این الگوریتم را می‌توان به صورت زیر بیان کرد:

پس از پیدا کردن خطای مینیمم، وزن های آموزش دیده شده در آن خطا، به عنوان وزن های شبکه ما برای آموزش شبکه شبکه، برای حل مسئله چینش انتخاب می‌شوند و در فاز دوم آموزش، گره ها یه عنوان ورودی به شبکه از پیش آموزش دیده شده با وزن‌های مشخص داده می‌شود و با استفاده از همان الگوریتم پس‌انتشار خطا، فرایند آموزش تا زمان مینیمم شدن خطا ادامه پیدا می‌کند. «شکل ۲-۳»

مراحل انجام این الگوریتم را می‌توان به صورت زیر بیان کرد:

---

[۱۰] Global Placement
[۱۱] Legalization
[۱۲] Detailed Placement
[۱۳] Back Propagation



شکل ۲-۲: فرایند آموزش شبکه برای پیدا کردن وزن‌های اولیه

برای پیدا کردن خطا هم به سادگی تابع $f(y, \hat{y}) = |\hat{y} - y|$ محاسبه شده است که در این رابطه $\hat{y}$ مقدار بدست آمده توسط شبکه است که در «شکل ۲-۲» با $\phi(x_i; w)$ نشان داده شده است.

در ادامه، نمودار زمانی بخش‌های مختلف الگوریتم مانند GP و LG و ... این الگوریتم در بستر CPU در دو حالت تک نخی و ۱۰ نخی برای طراحی bigblue4[۱۴] در «شکل ۲-۴» آورده شده است:

همانطور که انتظار می‌رفت، با افزایش تعداد نخ ها در CPU زمان انجام بخش‌های مختلف کاهش یافته است. برای نمونه زمان فاز GP در CPU ۱۰ نخی، ۱۰۶٫۱% کاهش پیدا کرده است.

آموزش فاز LG هم بر روی CPU تک، ۱۰، ۲۰ و ۴۰ نخی برای چندیت طراحی مختلف منجمله bigblue انجام شده است که خروجی‌های زمانی آن در «شکل ۲-۶» آورده شده است. همانطور که مشخص است به طور میانگین این فاز بر روی CPU تک نخی زیر ۱ دقیقه طول می‌کشد.

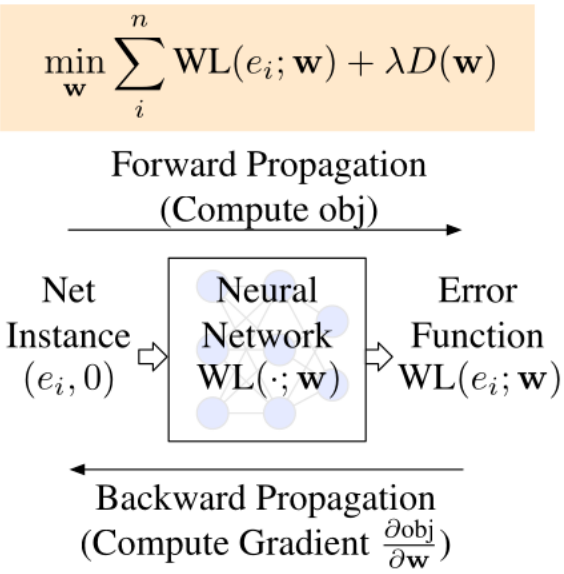
همانطور که قبلا هم بیان شد، زمان اجرای این الگوریتم به صورت خطی با افزایش تعداد سلول ها زیاد می‌شود. «شکل ۲-۷» نمودار رشد زمانی، با افزایش تعداد سلول ها را برای فاز GP نشان می‌دهد.

۲-۴ مزایا و معایب

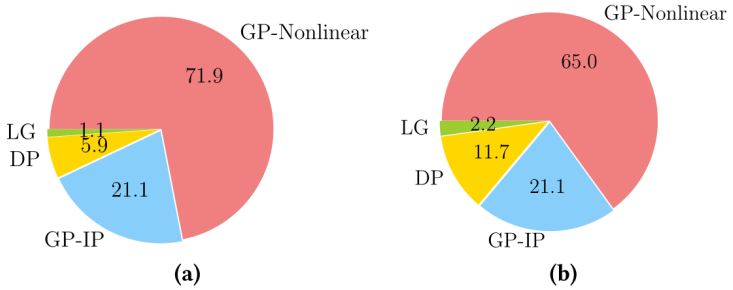
از مزایای روش پیشنهاد شده می‌توان به موارد زیر اشاره کرد:

- متن باز بودن آن

[۱۴] این طراحی متشکل از ۲ میلیون سلول است



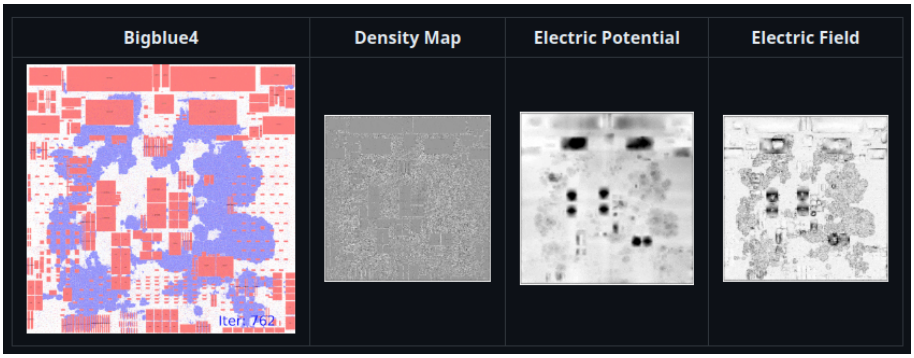
شکل ۲-۳: فرایند آموزش شبکه برای پیدا کردن بهترین چینش



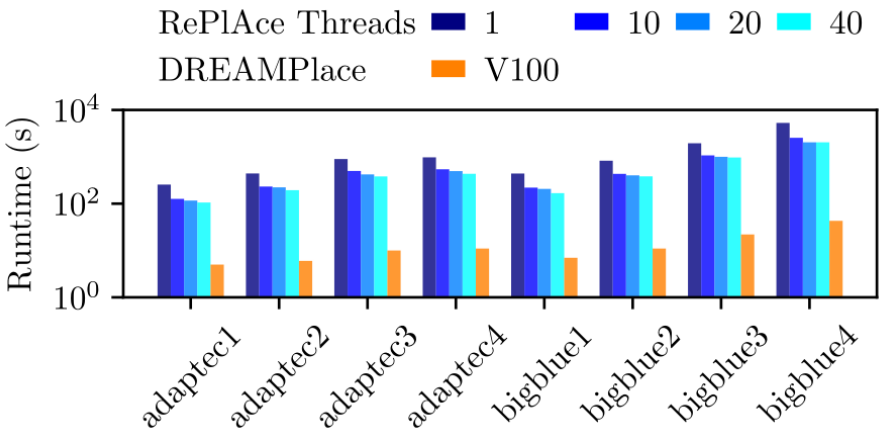
شکل ۲-۴: نمودار زمانی شبکه آموزش دیده شده در بستر CPU

- توسعه الگوریتم به دو زبان Python و C++
- توسعه الگوریتم بر دو بستر CPU و GPU
- هماهنگ بودن با سایر الگوریتم های تحلیلی
- سرعت بالای آن
- عدم افت کیفیت طراحی
- پشتیبانی[۱۵] قوی پروژه که توسط شرکت NVIDIA انجام می‌شود.
- و ...

[۱۵] Affiliation



شکل ۲-۵: خروجی واقعی شبکه برای مسئله چینش

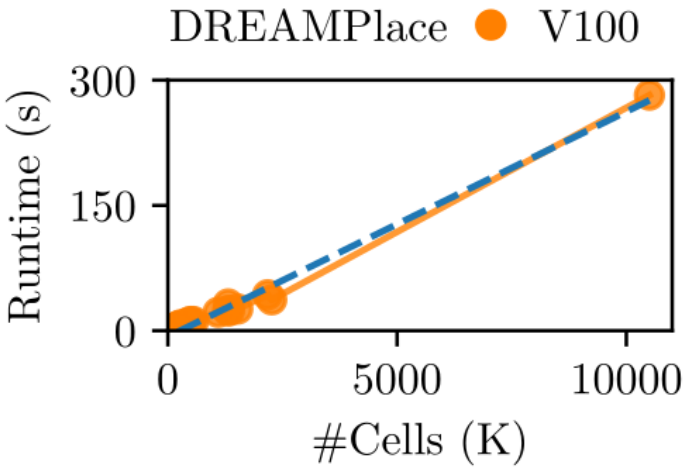


شکل ۲-۶: نمودار زمانی فاز LG بر بستر CPU

درکنار بیان مزایا می‌بایست به معایب پروژه را هم بیان کرد. در ادامه چند مورد از معایب پروژه انجام شده را بیان می‌کنیم:

- عدم اشاره مستقیم مقاله به ساختار شبکه‌عصبی استفاده شده
- نیازمند بودن به سیستمی قوی برای اجرای این الگوریتم
- عدم تست کردن الگوریتم با برچسب`Benchmark` های جدید تر مثل ISPD ۲۰۱۷ و ISPD ۲۰۱۸
- نیازمندی به پیشنیاز[۱۶] های مختلف
- عدم صحبت از توان مصرفی الگوریتم
- عدم ارائه گزارش از خطاهای ناشی از اجرای الگوریتم
- و ...

[۱۶] Dependency



شکل ۲-۷: تغییرات نمودار زمانی فاز GP

۲-۵ اجرای عملی الگوریتم

برای اجرای این الگوریتم بر روی سیستم شخصی می‌بایست مراحل زیر را طی کرد[۱۷]:

۲-۵-۱ نصب Git

با دستور زیر می‌توان گیت را نصب نمود:

```
$ sudo apt install git-all
```

همچنین با دستور زیر چک می‌کنیم که گیت به درستی نصب شده باشد:

```
$ git --version
```

اگر در خروجی ورژن گیت برگشت داده شود، یعنی نصب به درستی انجام شده است:

۲-۵-۲ دانلود مخزن الگوریتم

با استفاده از دستور زیر، مخزن[۱۸] الگوریتم را دانلود می‌کنیم.

```
$ git clone --recursive https://github.com/limbo018/DREAMPlace.git
```

پس از دانلود با دستور زیر به دایرکتوری فایل دانلود شده می‌رویم:

[۱۷] به دلیل استفاده اینجانب از سیستم‌عامل لینوکس، مراحلی که در ادامه نام برده شده است برای کاربران لینوکسی‌ست.
[۱۸] Repository

```
$ cd DREAMPlace
```

۲-۵-۳ نصب پیش‌نیاز ها

با دستور زیر، پیش‌نیاز ها[۱۹] ها را نصب می‌کنیم.

```
$ pip install -r requirements.txt
```

۲-۵-۴ بیلد نرم‌افزار

این نرم‌افزار را می‌توان به دو صورت بیلد کرد. استفاده از داکر[۲۰] و یا بیلد معمولی بر روی سیستم[۲۱].

```
$ mkdir build
$ cd build # we call this <build directory>
$ cmake .. -DCMAKE_INSTALL_PREFIX=<installation directory>
-DPython_EXECUTABLE=$(which python)
$ make
$ make install
```

بیلد نرم‌افزار، مدتی طول خواهد کشید. اگر بیلد به درستی انجام شود، پیغام `Building Successful` نمایش داده می‌شود.

۲-۵-۵ انتخاب بنچ‌مارک

با استفاده از دستور زیر می‌توان بنچ‌مارک مورد نظر را انتخاب کرد:

در این مقاله از بنچ‌مارک ISPD ۲۰۰۵ استفاده شده است.

```
$ python benchmarks/ispd2005_2015.py
```

---

[۱۹] Dependency

[۲۰] Docker

[۲۱] در این گزارش بیلد معمولی بر روی سیستم را توضیح می‌دهیم. برای بیلد بر روی داکر می‌توانید اینجا را ببینید.

۲-۵-۶ اجرای شبیه‌سازی

پس از انتخاب بنچ‌مارک به صورت زیر می‌توان برنامه را اجرا کرد:

```
$ cd <installation directory>
$ python unittest/ops/hpwl_unittest.py
```

فصل ۳

مقاله [۲]

۳-۱ ایده اصلی مقاله

ایده اصلی این مقاله نیز، حول محور مسئله مسیریابی در طراحی‌ها با استفاده از تکنیک‌های هوش مصنوعی قرار دارد با این تفاوت که در این مقاله مسیریابی ها مختص FPGA است و الگوریتم پیشنهاد شده برای مسیریابی بهتر درون FPGA پیشنهاد شده است. الگوریتم ارائه شده در این مقاله نیز متن‌باز است و مبتنی بر Python و کتابخانه PyTorch است و از موازی سازی در سطح GPU پشتیبانی می‌کند. در این مقاله ادعا شده است که طول مسیرها تا ۴٫۰ تا ۷٫۱۲ % کوتاه شده و سرعت مسیریابی نیز بیشتر از ۲ برابر روش‌های معمولی شده است.

۳-۲ کارهای پیشین

همانند مقاله قبل در این مقاله نیز اشاره شده است که روش‌های قدیمی که به روش‌های CAD[۱] معروف هستند در گذشته به طور عمده مورد استفاده قرار میگرفته است و مرحله مسیریابی به تنهایی ۴۱ تا ۸۶ درصد از کل زمان‌بندی مسیر طراحی CAD را به خود اختصاص می‌دهد [۹] و از این رو کاری بسیار زمان‌بر است. بنابر این کیفیت و کارایی الگوریتم های PAR[۲] بر طراحی بهینه بسیار تاثیر گذار هستند.

الگوریتم های بسیار زیادی مطرح شده است که همگی آنها مبتنی بر ابزار CAD ارائه شده توسط شرکت‌های سازنده FPGA هستند.

---

[۱] Computer-Aided Design

[۲] Placement and Routing

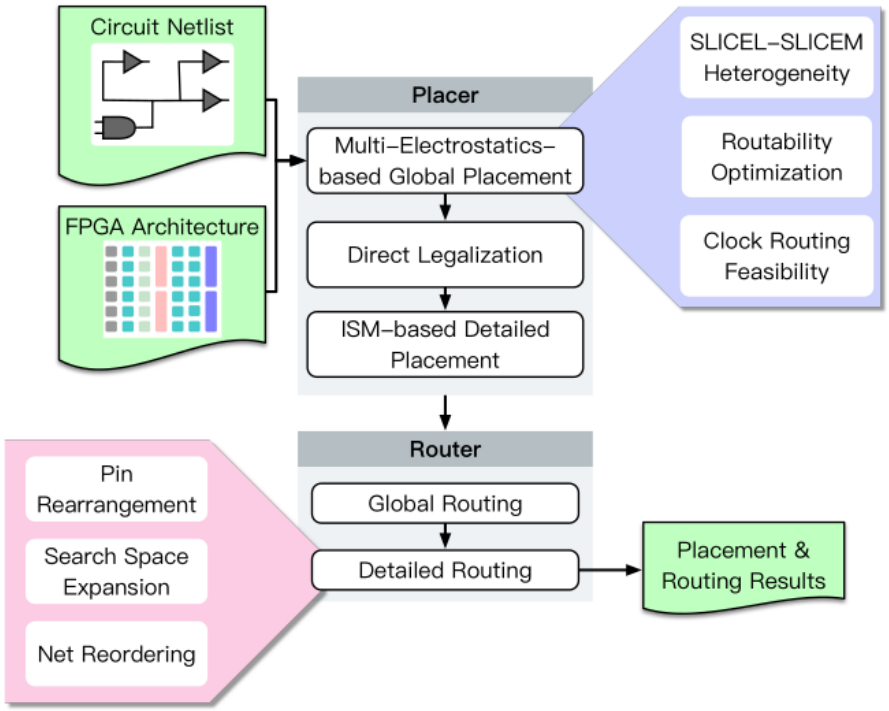
برای اینکه شرکت های تولید کننده FPGA از تولید مجدد ابزار CAD به خصوص، برای یک FPGA خاص جلوگیری کنند، ابزاری متن باز مبتنی بر کتابخانه یادگیری عمیق PyTorch ارائه شده است.

ایده های اصلی این مقاله را می‌توان به صورت زیر دسته بندی کرد:

- ابزار ارائه شده به نام OpenPARF است که ابزاری متن باز برای مسیریابی FPGA های پیشرفته، مبتنی بر یادگیری عمیق و قابل اجرا بر روی دو پلتفرم CPU و GPU است.
- این ابزار الگوریتم های مسیریابی غیر خطی را بر اساس یک میدان الکترواستاتیکی نا متقارن پیاده سازی می‌کند که قادر است به نتایج مکانیابی برتر تحت محدودیت های مختلف مسیریابی دست پیدا کند.

۳-۳ مراحل انجام الگوریتم

مراحل انجام این الگوریتم در «شکل ۳-۱» نشان داده شده است. همانطور مشخص است، این الگوریتم دارای ۲ مرحله چینش و مسیریابی است.



شکل ۳-۱: مراحل انجام الگوریتم OpenPARF

در ابتدا فایل‌های اطلاعت مکان‌ها «فایل هایی با فرمت «bookshelf و فایل های ساختار و معماری مسیریابی مورد نظر «فایل هایی با فرمت «XML در ساختار داده داخلی OpenPARF ساخته می‌شود و

در مرحله دوم، بر اساس ساختار داده داخلی، OpenPARF از الگوریتم های مکان‌یابی مبتنی بر میدان‌های الکترواستاتیکی چندگانه مسئله مسیریابی را در یک چارچوب بهینه‌سازی استاندارد استفاده می‌کند.

پس از انجام مسیریابی در مرحله اول، الگوریتم تلاش می‌کند با انجام مسیریابی های دقیق تر و با جزئیات بیشتر، مشکلات موجود در مسیر یابی اولیه انجام شده را بر طرف نماید.

پس این الگوریتم مسیر یابی از دو مرحله تشکیل می‌شود:

- مسیریابی اصلی سطح دانه درشت
- مسیر یابی دقیق و جزئی سطح دانه ریز

در مرحله اول یک مسیر اصلی و کلی پیشنهاد می‌شود و در مرحله دوم، مسیر یابی به صورت دقیق و منطقه به منطقه بررسی می‌شود و بهترین خروجی را تولید می‌کند.

۳-۴ مزایا و معایب

از مزایای روش پیشنهاد شده می‌توان به موارد زیر اشاره کرد:

- این مقاله نیز همانند قبلی یکی از مزیت های آن متن باز بودن آن است
- توسعه الگوریتم به دو زبان Python و C++
- توسعه الگوریتم بر دو بستر CPU و GPU
- ارائه نتایج برای دو بنچ‌مارک صنعتی و اکادمیک جدید
- مقایسه بسیار گسترده LUT ها و FF ها و سایر منابع مصرفی FPGA به هنگام استفاده از این الگوریتم.
- و ...

در کنار بیان مزایا می‌بایست به معایب پروژه را هم بیان کرد. در ادامه چند مورد از معایب پروژه انجام شده را بیان می‌کنیم:

- عدم اشاره مستقیم مقاله به ساختار شبکه‌عصبی استفاده شده در این الگوریتم

- نیازمند بودن به سیستمی قوی برای اجرای این الگوریتم. چرا که این الگوریتم بر روی سیستمی بسیار قوی آموزش داده شده است و به این نکته اشاره نشده است که مینیمم امکانات سیستم مورد نظر چه باید باشد که بتوان این الگوریتم را بر روی آن اجرا کرد.
- نیازمند بودن الگوریتم به FPGA های پیشرفته با منابع داخلی بالا. چرا که به نظر بنده این الگوریتم بر روی FPGA های قدیمی با منابع داخلی محدود قابل اجرا نمی‌باشد
- عدم صحبت از توان مصرفی الگوریتم
- عدم ارائه گزارش از خطاهای ناشی از اجرای الگوریتم
- و ...

۳-۵ اجرای عملی الگوریتم

۳-۵-۱ دانلود مخزن الگوریتم

با استفاده از دستور زیر، مخزن[۳] الگوریتم را دانلود می‌کنیم.

```
$ git clone https://github.com/PKU-IDEA/OpenPARF.git
```

۳-۵-۲ نصب وابستگی‌ها پیشنیازها

وابسگی‌های[۴] مورد نیاز را به صورت زیر نصب می‌کنیم:

```
  # * create and activate conda environment
$ mamba create --name openparf python=3.7
$ mamba activate openparf
# * common packages
$ mamba install cmake boost bison
# * Pytorch 1.7.1. Other version may also work, but have not been tested.
$ mamba install pytorch==1.7.1 torchvision==0.8.2 cudatoolkit=11.0 -c pytorch
```

[۳] Repository
[۴] Dependencies

```
# * python packages
$ pip install hummingbird-ml pyyaml networkx tqdm
```

۳-۶ بیلد نرم‌افزار

به صورت زیر، OpenPARF را بیلد می‌کنیم:

```
  $ mkdir build
$ cd build
$ cmake ../OpenPARF -DCMAKE_PREFIX_PATH=$CONDA_PREFIX
-DPYTHON_EXECUTABLE=$(which python)-DPython3_EXECUTABLE=$(which python)
-DCMAKE_INSTALL_PREFIX=<installation directory>
$ make -j8
$ make install
```

۳-۷ دانلود بنچ‌مارک ها

بنچ‌مارک های مورد استفاده در این مقاله را می‌توان از آدرس‌های زیر دانلود نمود:

1. ISPD 2016 FPGA Placement Benchmarks [download]
2. ISPD 2016 FPGA Placement Flexshelf Benchmarks [download]
3. ISPD 2017 FPGA Placement Benchmarks [download]
4. ISPD 2017 FPGA Placement Flexshelf Benchmarks [download]

می توان به صورت زیر نیز بنچ‌مارک ها را در ترمینال دانلود کرد:

```
  $ curl <url> --output <output filename>
```

۳-۸ لینک کردن بنچ‌مارک ها

با استفاده از دستورات زیر، می‌توان بنچ‌مارک های دانلود شده را لینک کرد:

```
$ ln -s <benchmark directory>/ispd2016 <installation
directory>/benchmarks/ispd2016
$ ln -s <benchmark directory>/ispd2016_flexshelf <installation
directory>/benchmarks/ispd2016_flexshelf
$ ln -s <benchmark directory>/ispd2017 <installation
directory>/benchmarks/ispd2017
$ ln -s <benchmark directory>/ispd2017_flexshelf <installation
directory>/benchmarks/ispd2017_flexshelf
```

۳-۹ اجرا کردن بنچ‌مارک ها

بنچ‌مارک ها را به صورت زیر می‌توان اجرا[۵] نمود:

```
$ cd <installation directory>
$ python openparf.py --config unittest/regression/ispd2016_flexshelf/
FPGA01_flexshelf.json
```

[۵] Run

فصل ۴

مقایسه مقاله های [۱، ۲] و نتیجه گیری

هر دو مقاله بررسی شده، به ارائه ساختاری متن‌باز جهت انجام دو فرایند مهم در طراحی به نام های چینش و مسیریابی با استفاده از تکنیک های یادگیری عمیق پرداخته است. با این تفاوت که در مقاله [۱] محدودیتی بر روی مدار یا ساختار مورد بخث گذاشته نشده است و می‌توان الگوریتم را برای هر مدار فشرده و خیلی فشرده «VLSI» ای آموزش داد و از آن در فرایند طراحی استفاده نمود. اما در مقاله [۲] الگوریتم ارائه شده فقط مخصوص چینش و مسیریابی FPGA هایی با منابع داخلی فراوان است. از این رو، کاربرد و عمومیت مقاله [۱] نسبت به مقاله [۲] بیشتر است.

هر دو مقاله برای انجام فرایند آموزش از سخت‌افزار های پیشرفته و گران قیمتی استفاده کرده اند.

در [۱] از سروری ۴۰ هسته لینوکسی با سی‌پی‌یو Intel E۵-۲۶۹۸ V۴ با فرکانس کاری ۲٫۲ گیگاهرتز و کارت‌گرافیک NVIDIA Tesla V۱۰۰ برای فرایند آموزش استفاده شده است.

و در [۲] از سروری لینوکس بیس که CPU آن Intel Xeon ۶۲۳۰ با فرکانس کاری ۱۰٫۲ گیگاهرتز و ۴۰ هسته هسته، با کارت گرافیک NVIDIA RTX ۲۰۸۰Ti و حافظه رم ۵۱۲ گیگابایتی استفاده شده است.

در هر دو مقاله سیستم های استفاده شده، سیستم های قوی و خاص منظوره ای هستند که کمتر در دسترس عموم مردم است. به همین دلیل ممکن است نتوان این الگوریتم ها را با سیستم‌های معمولی توسعه داد و این شاید به نوعی یکی از عیب‌های این دو الگوریتم به حساب آید.

در [۱] الگوریتم با دو بنچ‌کارک اکادمیک و صنعتی تست شده است که خروجی‌های آن به صورت زیر است: «شکل ۴-۱»

در [۲] نیز خروجی به ازای بنچ‌مارک ISPD ۲۰۱۷ ارائه شده است. «شکل ۴-۲»

همانطور که در « شکل۴-۱» مشاهده می‌شود، در الگوریتم DREAMPlace با افزایش تعداد سلول های

Table 2: Experimental results on ISPD 2005 benchmarks [25].

Design	#cells	#nets	RePlAce [6] (40 Threads)					DREAMPlace					
			HPWL	GP (s)	LG (s)	DP (s)	Total (s)	HPWL	GP (s)	LG (s)	DP (s)	IO (s)	Total (s)
adaptec1	211K	221K	73.26	106	5	24	139	73.30	5	0.5	25	5	37
adaptec2	255K	266K	81.87	194	7	30	236	82.19	6	0.5	32	6	47
adaptec3	452K	467K	193.20	382	22	53	466	194.12	10	1	58	10	82
adaptec4	496K	516K	175.23	434	21	61	524	174.43	11	2	65	11	92
bigblue1	278K	284K	89.85	168	3	30	206	89.43	7	0.3	35	6	51
bigblue2	558K	577K	138.09	383	22	90	505	136.69	11	9	90	12	125
bigblue3	1097K	1123K	304.83	967	46	138	1171	303.99	22	3	145	24	198
bigblue4	2177K	2230K	743.73	2037	56	329	2463	743.75	43	9	336	54	446
ratio	-	-	1.002	34.8	10.6	0.9	5.0	1.000	1.0	1.0	1.0	-	1.0

Table 3: Experimental results on industrial benchmarks.

Design	#cells	#nets	RePlAce [6] (40 Threads)					DREAMPlace					
			HPWL	GP (s)	LG (s)	DP (s)	Total (s)	HPWL	GP (s)	LG (s)	DP (s)	IO (s)	Total (s)
design1	1345K	1389K	340.42	974	46	155	1216	340.87	24	4	179	34	244
design2	1306K	1355K	275.46	1001	44	152	1238	275.76	24	5	180	32	244
design3	2265K	2276K	524.35	1767	84	257	2189	522.79	37	14	305	55	414
design4	1525K	1528K	455.22	1130	55	187	1424	454.38	26	8	207	37	281
design5	1316K	1364K	287.24	955	46	153	1193	288.41	22	4	186	33	248
design6	10504K	10747K	NA	>9100	NA	NA	NA	2356.88	282	73	1681	276	2323
ratio	-	-	1.000	43.1	9.5	0.9	5.0	1.000	1.0	1.0	1.0	-	1.0

شکل ۴-۱: خروجی های زمانی الگوریتم DREAMPlace برای دو بنچ‌مارک مختلف

Design	#LUT/#FF/#BRAM/#DSP	#Clock	RippleFPGA [13]			OpenPARF		
			PRT	RRT	RWL	PRT	RRT	RWL
CLK-FPGA01	211K/324K/164/75	32	278	10	238.54	131	10	205.44
CLK-FPGA02	230K/280K/236/112	35	250	15	261.85	127	14	246.65
CLK-FPGA03	410K/481K/850/395	57	537	24	648.69	206	24	594.00
CLK-FPGA04	309K/372K/467/224	44	346	18	440.09	157	19	420.30
CLK-FPGA05	393K/469K/798/150	56	501	25	560.18	201	23	510.62
CLK-FPGA06	425K/511K/872/420	58	545	28	678.43	218	28	617.28
CLK-FPGA07	254K/309K/313/149	38	288	13	276.29	136	13	256.62
CLK-FPGA08	212K/257K/161/75	32	235	10	213.06	119	10	196.67
CLK-FPGA09	231K/358K/236/112	35	312	13	297.02	148	14	250.98
CLK-FPGA10	327K/506K/542/255	47	465	23	544.07	195	14	451.28
CLK-FPGA11	300K/468K/454/224	44	421	24	516.67	182	30	421.52
CLK-FPGA12	277K/430K/389/187	41	378	18	403.59	167	20	336.03
CLK-FPGA13	339K/405K/570/262	47	393	21	464.78	178	19	428.41
Ratio			2.251	1.037	1.128	1.000	1.000	1.000

شکل ۴-۲: خروجی های زمانی الگوریتم OpenPARF برای بنچ‌مارک ISPD ۲۰۱۷

طراحی، زمان مراحل GP و LG و ... افزایش پیدا می‌کند و سلول ها تا ۲ میلیون و ۱۷۷ هزار عدد افزایش داده شده است که برای این مقدار سلول، مرحله GP ، ۴۳ ثانیه طول کشیده است.

برای الگوریتم OpenPARF هم هرچقدر دیزاین ها پیچیده تر می‌شود، هم تعداد منابع مصرفی داخلی FPGA بیشتر مصرف می‌شود و هم زمان‌بندی فاز چینش و مسیریابی نیز زیاد تر می‌شود.

نتیجه گیری ای که این‌جانب از این مطالعات دارم بدین صورت است:

با پیشرفت ابزارهای هوش مصنوعی، کم کم ابزارهای قدیمی طراحی دارند جای خود را به ابزارهای هوشمند می‌دهند. هرچند که همچنان راه زیادی وجود دارد تا جایگذینی کامل این ابزارها با هم اما بلاخره روزی فراخواهد رسید که قرایند طراحی از مرحله طراحی شماتیک تا Layout به صورت اتوماتیک و بدون دخالت انسان انجام

می‌شود.

این دو مقاله به پیاده‌سازی ۲ مرحله مهم از طراحی، یعنی چینش قطعات و مسیریابی آنها گام کوچ و موثری در تحقق این هدف برداشته اند.

امید است که بتوانیم با تحقیقات بیشتر و ارائه ساختارهای جدید به پیشرفت تکنولوژی در این مسیر کمک کنیم.

Bibliography

[1] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *Proceedings of the 56th Annual Design Automation Conference*, (117):1 – 6, 2019.

[2] J. Mai, J. Wang, Z. Di, G. Luo, Y. Liang, and Y. Lin. Openparf: An open-source placement and routing framework for large-scale heterogeneous fpgas with deep learning toolkit. *IEEE 15th International Conference on ASIC (ASICON)*, 2023.

[3] T. Lin, C. Chu, and G. Wu. Polar 3.0: An ultrafast global placement engine. *IEEE*, pages 520 – 527, 2015.

[4] A. Ludwin, V. Betz, and K. Padalia. High-quality, deterministic parallel placement for fpgas on commodity hardware. *ACM*, pages 14 – 23, 2008.

[5] W. Li, M. Li, J. Wang, , and D. Z. Pan. Utplacef 3.0: A parallelization framework for modern fpga global placement. *ACM*, pages 922 – 928, 2017.

[6] J. Cong and Y. Zou. Parallel multi-level analytical global placement on graphics processing units. *ACM*, pages 681 – 688, 2009.

[7] C. Lin and M. D. Wong. Accelerate analytical placement with gpu: A generic approach. *IEEE*, pages 1345 – 1350, 2018.

[8] C. Lin and M. D. Wong. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE TCAD*, 27(7):1228 – 1240, 2008.

[9] K. E. Murray, S. Whitty, S. Liu, J. Luu, , and V. Betz. Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial cad. *ACM TRETS*, pages 1 – 18, 2015.

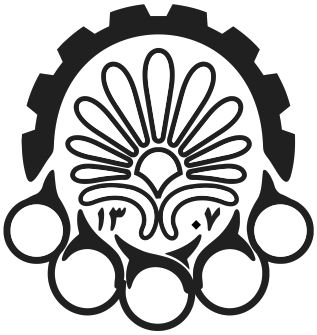
[10] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, , and R. Aggarwal. Clock-aware fpga placement contest. *ISPD*, pages 159 – 164, 2017.

Abstract

The placement of components and routing of integrated circuits on very large scales has always been one of the most challenging stages in the circuit design process. In the past, these two stages were manually performed by human operators. This manual process often led to repeated designs due to certain technical considerations, requiring the redesign and rerouting of the circuit, which consumed a considerable amount of time. As circuit designs have grown in complexity in modern integrated circuits, this process has become increasingly difficult and perhaps even seemingly impossible. Technological advancements in the field of artificial intelligence and deep learning have brought about changes in the world of integrated circuit design. Today, the world of IC design is moving towards a direction where the placement and routing of ICs can be done without human intervention, entirely automatically, with high precision and very low error rates.

In this report, we will examine and discuss the advantages and disadvantages of two similar methods proposed in the articles [1] and [2] for achieving this.

Keywords: Placement, Routing, Deep Learning



Amirkabir University of Technology

(Tehran Polytechnic)

Department of Computer Engineering

Advanced VLSI Final Research Report

Review and comparison of deep learning based methods for placement and routing problem for VLSI circuits

By:

Reza Adinepour

Instructor:

Prof. Sedighi

February 2024