بهار ۱۴۰۰، دوره ۶، شماره ۱، پیاپی ۲۰
DOR :20.1001.1.2538161.1400.6.1.2.0

دریافت مقاله: ۱۳۹۹/۱۰/۰۱
پذیرش مقاله: ۱۴۰۰/۰۱/۲۹

نوع مقاله: پژوهشی

شتاب‌دهنده دارای قابلیت بازپیکربندی برای اجرای کارآمد شبکه‌های عصبی

پریا دربانی
دانشجوی دکترا، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران
پست الکترونیکی: paria_darbani@cmps2.iust.ac.ir

نظام رهبانی
پژوهشگر پسادکترا، پژوهشکده علوم کامپیوتر، پژوهشگاه دانش‌های بنیادی، تهران، ایران
پست الکترونیکی: rohbani@ipm.ir

حاکم بیت‌الهی*
استادیار، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران
پست الکترونیکی: beitollahi@iust.ac.ir

پژمان لطفی کامران
دانشیار، پژوهشکده علوم کامپیوتر، پژوهشگاه دانش‌های بنیادی، تهران، ایران
پست الکترونیکی: plotfi@ipm.ir

چکیده

شبکه‌های عصبی عمیق[1] به‌طور گسترده در کاربردهای هوش مصنوعی استفاده می‌شوند. انتقالات زیاد داده‌ها و تعداد محاسبات فراوان از ویژگی‌های اجرای این شبکه‌هاست. شبکه‌های عصبی از چندین لایه تشکیل شده‌اند که هر لایه نسبت به دیگر لایه‌ها اندازه منحصربه‌فرد و متفاوتی دارد. معمولاً ابعاد شتاب‌دهنده[2] بر مبنای میانگینی از اندازهٔ دسته‌ای از لایه‌ها، تعیین و ساخته می‌شود. هنگام اجرای برخی از لایه‌ها به دلیل عدم همپوشانی کامل ابعاد آن لایه با ابعاد پردازنده، تعدادی از منابع بی‌استفاده می‌مانند؛ یعنی با وجود نیاز به منابع محاسباتی بیشتر برای اجرای سریع‌تر محاسبات، اختلاف اندازه برخی لایه‌های شبکه عصبی با ابعاد شتاب‌دهنده، مانعِ دستیابی به حداکثر کارایی می‌شود. معماری پیشنهادی با استفاده از قابلیت بازپیکربندی امکان تغییر ابعاد پردازنده و نزدیک شدن به ابعاد لایهٔ در حال اجرا را فراهم می‌کند. این معماری مشکل بی‌استفاده ماندن منابع را بهبود داده و در برخی موارد کاملاً برطرف می‌کند. بهبود بهره‌وری، سرعت محاسبات مدل گوگل‌نت را به‌طور متوسط ۲۷/۴۱٪ افزایش داده و دفعات خواندن از حافظه داخلی را حدود ۲۳٪ نسبت به معماری پایه کاهش داده است. بهبودها در حالی است که سربار سخت‌افزاری بسیار کم و قابل

* نویسندهٔ مسئول

1- Deep Neural Network
2- Accelerator

چشم‌پوشی به سیستم اعمال شده است.

واژه‌های کلیدی: شبکه عصبی عمیق، شتاب‌دهنده، مدل یادگیری ماشین، منابع بی‌استفاده، معماری بازپیکربند.

۱- مقدمه

امروزه به‌طور گسترده از کاربردهای گوناگون هوش مصنوعی در فنّاوری استفاده می‌شود. به‌عنوان مثال پردازش تصویر و تشخیص صدا در تلفن‌های هوشمند، اتومبیل‌ها، دستگاه‌های پزشکی و ... کاربرد دارند. شبکه‌های عصبی عمیق نتایج مورد انتظار را با دقت خوبی فراهم می‌کنند اما نیازمند محاسبات بسیار زیاد و ارتباطات فراوان با حافظه هستند [۱]. برای مثال شبکه الکس‌نت [۲] که از شبکه‌های عصبی کوچک محسوب می‌شود، برای پردازش یک عکس با ابعاد ۲۲۷ در ۲۲۷ بیش از ۷۲٤ میلیون عملیات ضرب انجام می‌دهد و بیش از دو میلیارد بار به حافظه مراجعه می‌کند.

لایه‌های شبکه عصبی نسبت به هم وابستگی داده‌ای[۳] دارند. به همین دلیل لایه‌ها یکی پس از دیگری اجرا می‌شوند. محاسبات هر لایه می‌توانند به‌صورت هم‌زمان انجام شوند ولی محدودیت‌های سخت‌افزاری باعث می‌شود که فراهم کردن بستری برای انجام همه محاسبات یک لایه به‌صورت هم‌زمان به‌صرفه نباشد. ازاین‌رو بنابر ظرفیت منابع محاسباتی موجود، داده‌ها دسته‌دسته از حافظه فراخوانی و سپس پردازش می‌شوند تا نتیجه نهایی حاصل شود. با توجه به کاربرد فراوان شبکه‌های عصبی و محدودیت‌های سخت‌افزاری، روش‌هایی که منجر به اجرای کارآمد و بهبود مصرف انرژی می‌شوند برای سامانه‌های هوش مصنوعی حیاتی هستند.

شبکه‌های عصبی بر روی بُن‌سازه[٤]های مختلف، مثل پردازنده‌های معمولی[٥]، پردازنده گرافیکی[٦]، FPGA و معماری‌های سفارشی قابل اجرا هستند که هرکدام از این بُن‌سازه‌ها نسبت به دیگری دارای مزایا و معایبی است. توان گذردهی[۷] پردازنده‌های معمولی ضعیف است و پردازنده گرافیکی انرژی زیادی مصرف می‌کند. FPGA از انعطاف[۸] بالایی در مقابل اندازه‌های مختلف لایه‌ها برخوردار است ولی نسبتاً زمان زیادی صرف برنامه‌ریزی و پیکربندی[۹] آن می‌شود. با وجود هزینه بالای طراحی و ساخت، شتاب‌دهنده سفارشی به دلیل کارایی بالا در هر وات[۱۰] و مساحت کمترش، گزینه مناسبی است.

شتاب‌دهنده آرایه‌ای که از دستهٔ معماری سفارشی است، ساختار کارآمد و شناخته‌شده‌ای برای اجرای شبکه‌های عصبی عمیق است [۳]. این معماری شامل آرایه‌ای از عناصر پردازشی متصل به هم است که هم‌زمان با هم کار می‌کنند و گذردهی بالایی را فراهم می‌کنند [٦ – ٤]. اندازه ابعاد شتاب‌دهنده‌های پیشرفته[۱۱] مانند لاکُنیک[۱۲] [٤]، TPU [۷] و آیریس [۸] بر اساس میانگینی از اندازه‌های لایه‌های شبکه عصبی تعیین می‌شوند. به همین دلیل بعضی از لایه‌ها همپوشانی کامل با آرایه‌ای از عناصر پردازشی ندارند [۹, ٥]. برای مثال، کوچک‌تر بودن حداقل یکی از ابعاد لایه شبکه عصبی نسبت به بُعد متناظرش در شتاب‌دهنده، منجر به بی‌استفاده ماندن تعدادی از عناصر پردازشی می‌شود. از طرفی اگر بُعد دیگر لایه بزرگ‌تر از بُعد متناظر شتاب‌دهنده باشد، عملیات خواندن از حافظه و پردازش داده باید بارها تکرار[۱۳] شود تا محاسبه لایه پایان پذیرد. این بدان معنی است که با وجود تعدادی عناصر پردازشی بی‌استفاده در هر تکرار، برای جبران کمبود عناصر پردازشی در بُعدِ دیگر، شتاب‌دهنده مجبور به تکرار چندباره عملیات می‌شود. درنتیجه عدم استفاده بهینه از تمام منابع محاسباتی منجر به چرخه[۱٤]های محاسباتی ناکارآمد، افزایش زمان اجرا و مصرف توان بیشتر می‌شود [۱۱, ۱۰, ٥].

مشکل منابع بی‌استفاده به‌طور گسترده در

3- Data-dependency
4- Platform
5- CPU
6-GPU
7- Throughput
8- Flexibility
9- Configuration
10- Performance per watt
11- State-of-the-art
12- Laconic
13- Iteration
14- Cycle

شـــتاب‌دهنده‌ها وجود دارد [۳] و مانع دســـتیابی آن‌ها به حداکثر کارایی ممکن می‌شـــود [۴, ۷, ۱۰]. شبیه‌ســـازی‌ها نشان می‌دهد که شتاب‌دهنده‌های آرایه‌ای به‌طور متوسط حدود ۴۴٪ تا ۵۷٪ از ظرفیت منابع موجودشـــان اســـتفاده می‌کنند.

اتصالات ثابت در معماری آرایه‌ای مانع از دســـت‌یابی به یک ســـاختار جامع و کارا برای تمام لایه‌های شبکه‌های عصبی عمیق بـــا ابعاد مختلف می‌شـــود [۹, ۱۲, ۱۳, ۱۴]. وجود انعطاف در این ساختار می‌تواند استفاده از عناصر پردازشی را بهینه کند و به افزایش کارایی کمک کند. ما یک ســـاختار آرایه‌ای را معرفی کردیم که در آن شتاب‌دهنده بتواند انـــدازه ابعاد خود را تا جای ممکن بـــه اندازه ابعاد لایهٔ آمادهٔ اجرا نزدیک کند. ســـنجش ســـاختار پیشنهادی به‌وسیلهٔ یک شبیه‌ساز که بر پایه[15] چرخه‌های محاسباتی ایجاد شده، انجام شـــد. معماری پیشنهادی توانسته برای بعضی لایه‌های گوگل‌نت [۱۵] میزان اســـتفاده از منابع را در مقایسه با معماری پایه ۱۰۰٪ بهبود دهد. در پی استفاده بهتر از منابع، سرعت محاسبات افزایش پیدا کرده و دفعات دسترسی به حافظه داخلی نیز کاهش یافته است.

۱-۱- کارهای پیشین

شـــتاب‌دهنده‌های آرایـــه‌ای بـــه دلیـــل کارایـــی بالا و پیاده‌سازی نسبتاً آسان شناخته شده هستند. دادیانائو [۱۶] ساختار پایه دسته‌ای از معماری‌های آرایه‌ای تکرارشونده است که برای معماری پیشنهادی نیز به‌عنوان ساختار پایه در نظر گرفته شده است. دادیانائو نسبت به پردازنده‌های گرافیکی تا ۳۳۰ برابر انرژی کمتری مصرف می‌کند [۱۴]. این ساختار از ۱۶ کاشی[16] تشکیل شده که هرکدام از آن‌ها حاوی ۱۶ عنصر پردازشی هســـتند. هر عنصر پردازشی حاوی ۱۶ ضرب‌کننده اســـت. مراحـــل اجرای این معماری به‌صورت خلاصه به این ترتیب اســـت: ۱) اطلاعات عکس اولیه و وزن‌های لایه نخســـت از حافظـــه خارجی خوانده می‌شـــود. ۲) وزن‌ها درون میانگیرهـــای وزنی مربوط به هر کاشی توزیع می‌شـــوند. اطلاعات عکس درون میانگیر مخصـــوص داده ورودی ثبت می‌شـــود. ۳) در هر چرخه ۱۶ کانال ورودی به همه واحدهای پردازشـــی هم‌پخشی[17] می‌شود. هر عنصر پردازشی ۱۶ کانال ورودی و ۱۶ کانال وزنی از کانال‌های یک پالایه را از میانگیر می‌خواند. در هم ضرب می‌کند. نتایج ضرب‌ها به‌وسیله درخت تجمیعی[18] با هم جمع می‌شـــوند. سپس یک خروجی جزئی[19] در میانگیر انباشتگر[20] ذخیره می‌شـــود. تا پایان محاسبه یک لایه این روند تکرار می‌شود. ۴) بعد از اتمام محاسبات لایه، نتیجه به تابع فعال‌ساز[21] می‌رود تا نتیجه نهایی تولید شود. ۵) در انتها اطلاعات از میانگیر خروجی به حافظه منتقل می‌شود و لایه بعدی به‌عنوان ورودی آن را فراخوانی می‌کند.

در این معماری هر ردیف از عناصر پردازشـــی در هر کاشـــی مســـئول اجرای یک پالایه از وزن است. اگر تعداد پالایه‌ها بیش از تعداد ردیف‌ها باشد پس از اتمام محاسبات یک پالایـــه، پالایهٔ بعدی به ردیف اختصـــاص می‌یابد. اگر تعداد کانال‌های یک پالایه بیشتر از تعداد ضرب‌کننده‌های یک عنصر پردازشـــی باشـــد، عملیات پردازش در چندین چرخه تکرار می‌شود تا پردازش تمام کانال‌های یک پالایه تکمیل شـــود. پس تعداد پالایه‌های وزن هـــر لایه از مدلی که قرار اســـت به‌وســـیله دادیانائو اجرا شود باید مضربی از ۲۵۶ (تعداد کاشـــی‌ها × تعداد عناصر پردازشـــی در هر کاشی) باشـــد تا تمام عناصر پردازشـــی به‌طور کامل به کار گرفته شـــوند و عنصر پردازشـــی بیکاری باقی نماند؛ امـــا در واقعیت مدلی بـــا این ویژگی وجود نـــدارد [۱۴] و همه وزن‌هـــای لایه‌های یک مدل مضرب صحیحی از ۲۵۶ نیســـتند. برای مثال تعداد پالایه‌هـــای ۵۰٪ از لایه‌های مدل الکس‌نت [۲] مضرب ۲۵۶ نیستند. مسئله بی‌استفاده ماندن منابع در تمام شـــتاب‌دهنده‌ها و از جمله ساختار آرایه‌ای تکرارپذیر وجـــود دارد [۵]. راه‌حل‌هایی بـــرای بهبود این مشکل در سطح معماری ارائه شده است.

15- Baseline architecture
16- Tile
17- Broadcast
18- Reduction-tree
19- Partial output
20- Accumulator buffer
21- Activation function

استفاده از ابعاد کوچک‌تر: لاکُنیک [٤]، شِیپ‌شیفتر [٦] و استریپس [١٤] از پردازش در سطح بیت برای محاسبات مدل‌های شبکه عصبی استفاده می‌کنند و به این ترتیب به جای استفاده از ضرب‌کننده‌های بزرگ، می‌تواند از دروازه[22]های تک‌بیتی استفاده کند. پراگماتیک [١٧] در کنار استفاده از پردازش بیتی، برای افزایش انعطاف‌پذیری عناصر پردازشی در ستون‌های مجاور را به هم متصل می‌کند. اما این کار بهره‌وری را فقط در لایه‌هایی که دارای تعداد کانال‌های کم هستند افزایش می‌دهد. شیدیانائو [١٠] با کوچک کردن ابعاد آرایه‌ای از عناصر پردازشی، احتمال همپوشانی لایه با آرایه و بهره‌وری را افزایش داده است اما مانند معماری‌های قبلی این روش نیز باعث بدتر شدن گذردهی شده است. طوری که بالاترین کارایی گزارش شده از این معماری ١٩٤ GOP/S است [٥].

مدیریت نمایه‌ها: اسنپ [١٨] و استیچ [١٩] برای استفاده بهینه از منابع به نگهداری و مدیریت نمایهٔ[23] ورودی‌ها و وزن‌ها در یک واحد سخت‌افزاری جداگانه می‌پردازند. اگر معادله‌ای تأثیر درنتیجه نهایی نداشته باشد، نمایه مربوط به عملگرها غیرفعال می‌شود و با این کار از انجام محاسبات اضافی جلوگیری می‌شود. اما به دلیل اختلاف اندازه‌های وزن‌ها و ورودی‌ها روش‌هایی مثل هَرَس[24] کردن خود می‌تواند باعث بی‌استفاده ماندن بیشتر منابع شود [٢٠]. این روش باعث افزایش سربار مساحتی و مصرف توان می‌شود در حالی که مسئله را به‌طور مؤثر حل نکرده است.

اتصالات با قابلیت بازپیکربندی: آیریس [٨] یک ساختار با قابلیت بازپیکربندی است که از روش‌های استفاده مجدد از داده[25] با هزینه سخت‌افزاری زیاد استفاده می‌کند. آیریس در توزیع وزن‌ها دارای ضعف است و گاهی حدود ٥٠٪ از زمان اجرایش صرف انتقال داده می‌شود. TPU [٧] یک آرایه از عناصر پردازشی با ابعاد ١٢٨ در ١٢٨ و اتصالاتِ سختِ بینشان است که ویژگی مهم آن استفاده مجدد از داده است. سیگما [٣] برای حل مشکل منابع بی‌استفاده در TPU از ساختار مائری [٩] الهام گرفته و درخت جمع‌کننده‌ای به TPU اضافه کرده است. به این ترتیب انعطاف TPU بیشتر شده ولی سربار زیادی را به سیستم تحمیل کرده است. شبیه‌سازی‌ها نشان داد که سربار سخت‌افزاری این معماری حداقل سه برابر بیشتر از معماری پیشنهادی است.

با وجود تمام تلاش‌ها، مسئله بی‌استفاده ماندن منابع همچنان در شتاب‌دهنده‌ها وجود دارد. با توجه به نیاز روزافزون استفاده از شبکه‌های عصبی در تمام ابعاد فنّاوری، روشی که بدون سربار سخت‌افزاری زیاد بتواند این مشکل را بهبود دهد کارآمد خواهد بود.

٢- معماری پیشنهادی

نیاز به نگاشت[26] میلیون‌ها پارامترِ مدل‌های شبکه عصبی بر روی منابع محدود حسابی و حافظه‌ای، منجر به بروز تکرارهای پی‌درپی عملیات و استفاده مکرر از منابع موجود تا رسیدن به نتیجه می‌شود [٩]. اندازه ابعاد شتاب‌دهنده‌ها ثابت است ولی لایه‌ها اندازه‌های گوناگونی دارند. درنتیجه با کوچک‌تر بودن حداقل یکی از ابعاد لایه، مشکل عدم استفاده بهینه از منابع موجود پدید می‌آید و می‌تواند منجر به تکرارهای بیشتر برای تکمیل پردازش شود [٢١, ١٨, ٧, ٤]. نسبت استفادهٔ بهینه از منابع در هر معماری و هر شبکه عصبی میزان متفاوتی است. معماری کانونت [٢٢] هنگام اجرای مدل VGG [٢٣] فقط از ٥٥٪ درصد از منابعش استفاده می‌کند. معماری SCNN [١٢] یک‌پنجم از منابعش را به کار می‌گیرد [١١]. FPGAها وقتی در حالت تک‌پردازنده، مدل الکس‌نت را اجرا می‌کنند حدود ٢٤٪ از منابعشان استفاده می‌شوند [٢١]. میزان استفاده در بعضی موارد از این هم بدتر است، مثلاً در یک مورد فقط ٤٪ استفاده از منابع TPU گزارش شده است [٩, ٣].

زمان اجرای محاسبات در شتاب‌دهندهٔ آرایه‌ای با فرمول (١) تناسب دارد:

22- Gate
23- Index
24- Pruning
25- Data reusability
26- Mapping

(۱)

$$\text{چرخه‌های محاسباتی} \approx \left\lceil \frac{\text{تعداد کانال‌های لایه}}{\text{تعداد ضرب‌کننده‌های هر عنصر پردازشی}} \right\rceil \times \left\lceil \frac{\text{تعداد پالایه‌های لایه}}{\text{تعداد کاشی‌ها} \times \text{تعداد ردیف‌ها در هر کاشی}} \right\rceil$$

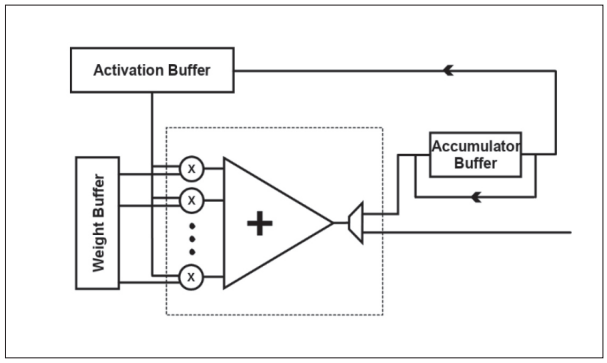
در فرمول (۱) صورت کسرها مربوط به مشخصات مدل شبکه عصبی است و مخرج کسرها مربوط به مشخصات شتاب‌دهنده‌ای که آن را اجرا می‌کند. با توجه به این معادله برای کم کردن چرخه‌های محاسباتی مطلوب است که حاصل هرکدام از کسرها تا جای ممکن به عدد ۱ نزدیک شود. درنتیجه بهتر است که صورت و مخرج هر کسر تا جای ممکن به هم نزدیک‌تر شوند. (قابل ذکر است که برای اختصار و طرح موضوع شفاف، محاسبات مربوط به ورودی‌ها در فرمول (۱) و در این مقاله مطرح نشده است.) مشخصات مدل شبکه عصبی که در صورت کسرها آمده ثابت و غیرقابل تغییر است ولی اگر بتوان معماری را طوری طراحی کرد که تعداد ردیف‌های شتاب‌دهنده نزدیک به تعداد پالایه‌های لایه باشد و همچنین تعداد ضرب‌کننده‌های هر عنصر پردازشی نزدیک به تعداد کانال‌های درون هر پالایه باشد؛ آنگاه بهره‌وری افزایش می‌یابد و در پی آن از تعداد چرخه‌های محاسباتی کاسته می‌شود. این همان ایده‌ای است که معماری پیشنهادی دنبال می‌کند.

برای رسیدن به این مقصود یک شتاب‌دهنده آرایه‌ای را پیشنهاد داده‌ایم که به‌وسیله واحد بازپیکربندی که به هر ستون از کاشی‌های این معماری متصل است، امکان تغییر اندازه عناصر پردازشی (تغییر تعداد ضرب‌کننده‌ها در هر عنصر پردازشی) و تغییر تعداد ردیف‌ها (تعداد عناصر پردازشی در هر ستون) را فراهم می‌کند. این تغییرات به این معنی است که معماری پیشنهادی می‌تواند اندازه ابعاد خود را بر اندازه ابعاد لایهٔ آمادهٔ اجرا منطبق یا به آن نزدیک کند. معماری پیشنهادی توانسته است کارایی اجرای مدل گوگل‌نت را ۲۷/۵٪ نسبت به شتاب‌دهنده پایه بهبود بخشد. ساختار پیشنهادی دارای یک واحد مدیریت برای زمان‌بندی عملیات است که سیگنال‌های لازم را برای خواندن یا نوشتن از/ به حافظه تولید می‌کند؛ اندازه بهینه عناصر پردازشی و تعداد ردیف‌ها را تعیین می‌کند و پیکربندی را به وسیلهٔ برنامه‌ریزی حافظه‌های کوتاه‌مدت یک‌بیتی با دسترسی تصادفی[27] انجام می‌دهد.

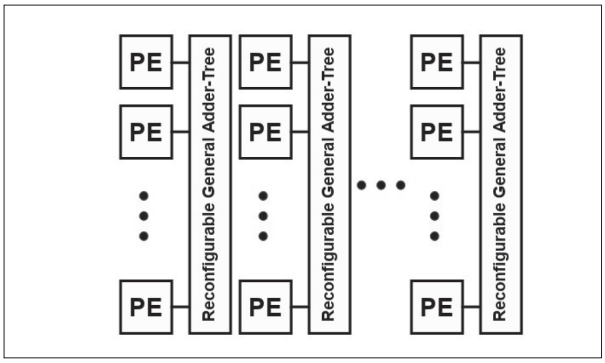
ساختار عنصر پردازشی پیشنهادی در شکل (۱) به‌وسیلهٔ نقطه‌چین محدود شده است. نتیجه محاسبات هر عنصر پردازشی می‌تواند در میانگیر انباشتگر ذخیره گردد یا به خروجی مستقیم فرستاده شود. واحد مدیریت بازپیکربندی برای انتخاب یکی از این دو حالت محاسباتی بر اساس معادلاتی شبیه فرمول (۱) انجام می‌دهد و بنابر مقایسه نتایج تصمیم می‌گیرد. سپس با برنامه‌ریزی واتسهیم‌گر[28] ۱ به ۲ آن را اجرا می کند. شکل (۲) ساختار کاشی پیشنهادی را نشان می‌دهد. همان‌طور که دیده می‌شود، عناصر پردازشی در هر ستون به‌وسیله یک درخت تجمیعی بازپیکربند به یکدیگر متصل هستند. با این حساب اگر کاشی دارای ۱۶ ستون باشد، ۱۶ عدد درخت تجمیعی بازپیکربند در کاشی وجود دارد. درخت تجمیعی بازپیکربند امکان ممزوج شدن عناصر پردازشی یک ستون را می‌دهد.

ساختار هر درخت تجمیعی بازپیکربند در شکل (۳) نمایش داده شده است. اگر هر ستون ۱۶ عنصر پردازشی داشته باشد، درخت تجمیعی بازپیکربند مطابق شکل (۳) دارای ٤ سطح خواهد بود. گره‌های درخت تجمیعی بازپیکربند جمع‌کننده هستند. هرکدام از گره‌های سطح اول یک جفت از عناصر پردازشی یک ستون از کاشی را به هم متصل می‌کنند. گره‌های سطوح دوم و سوم علاوه بر جمع‌کننده، میزبان یک واتسهیم‌گر ۱ به ۲ نیز هستند. این حافظه‌های کوتاه مدّت یک بیتی با دسترسی تصادفی برای انتخاب راه خروج، بین سطح بعدی درخت یا میانگیر متصل به گره استفاده می‌شوند. گره ریشه هنگامی استفاده می‌شود که بخواهیم ۱۶ عنصر پردازشی اولیه که هرکدام ۱۶ ضرب‌کننده دارند، نقش یک عنصر پردازشی با ۲۵۶ ضرب‌کننده را بازی کنند.

27- 1-bit SRAM
28- De-multiplexer



شکل ۱: ساختار عنصر پردازشی پیشنهادی

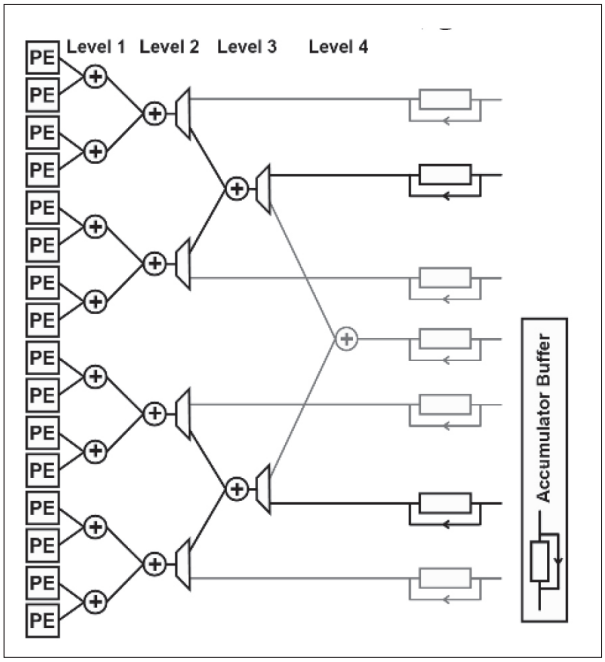


شکل ۲: ساختار کاشی در معماری پیشنهادی

لایه ششم از مدل گوگل‌نت دارای ۱۲۸ فیلتر و ۹۶ کانال است. اگر این مقادیر را در فرمول (۱) جایگذاری کنیم و محاسبات را با پیکربندی‌های مختلف و ممکنِ درخت تجمیعی بازپیکربند انجام دهیم، به این نتیجه می‌رسیم که عناصر پردازشی با ۱۲۸ ضرب‌کننده و کاشی‌هایی با دو ردیف (دو عنصر پردازشی در هر ستون) برای محاسبهٔ لایه ششم بهینه خواهد بود.

طبق فرمول (1) $\lceil (128) / (2 \times 16) \rceil \times \lceil (96) / (128) \rceil$ کوچک‌تر است از $\lceil (16) / (128) \rceil \times \lceil (96) / (16 \times 16) \rceil$. واحد مدیریت بازپیکربندی محاسباتی مشابه را انجام می‌دهد و بر اساس مقایسه نتایج، پیکربندی مطلوب را انتخاب می‌کند. سپس با برنامه‌ریزی واتسهیم‌گرهای داخل عنصر پردازشی و درخت تجمیعی بازپیکربند، پیکربندی مورد نظر را قبل از اجرای لایه پیاده‌سازی می‌کند. این پیکربندی تا اتمام محاسبات لایه ثابت است. شکل (۳) مثالی را نشان می‌دهد که در آن درخت تجمیعی بازپیکربند طوری برنامه‌ریزی شده که هر ۸ تا از عناصر پردازشی بتوانند به هم متصل شوند و مانند یک عنصر پردازشی با ۱۲۸ ورودی عمل کنند. در شکل (۳) گره‌ها و شاخه‌های فعال با رنگ سیاه و بخش‌های غیرفعال با رنگ خاکستری نمایش داده شده‌اند.

به نظر می‌رسد که هر چه عناصر پردازشی اولیه دارای ورودی‌های کوچک‌تری باشند و درخت تجمیعی بازپیکربندِ بزرگ‌تری داشته باشیم، انعطاف‌پذیری شتاب‌دهنده بیشتر می‌شود و امکان بهبود بیشتری را فراهم می‌کند؛ اما با



شکل ۳: ساختار درخت تجمیعی بازپیکربند در هر ستون برای معماری با ۱۶ ردیف در هر کاشی

توجه به سربار سخت‌افزاری باید نقطه تعادلی برای تعیین اندازه‌ها در نظر گرفته شود. بر این اساس پژوهشی بر روی چند مدل شبکه عصبی انجام شد که نتایج آن در جدول (۱) آمده است. این جدول پیکربندی‌های گوناگون یک کاشی با ابعاد ۱۶ ردیف در ۱۶ ستون و هر عنصر پردازشی شامل ۱۶ ضرب‌کننده را نمایش می‌دهد. بررسی‌ها بر روی چندین مدل نشان دادند که -طبق ردیف اول جدول (۱)- استفاده از عناصر پردازشی با ۲۵۶ ورودی، بیشتر از باقی پیکربندی‌ها مطلوب لایه‌ها قرار گرفته است (برای ۵۰٪ لایه‌ها این پیکربندی بهینه است).

همچنین این جدول نشان می‌دهد که عناصر پردازشی

جدول ۱: حالت‌های مختلف پیکربندی کاشی با ابعاد ۱۶ ردیف در ۱۶ ستون و هر عنصر پردازشی حاوی ۱۶ ضرب‌کننده

	عناصر پردازشی در هر کاشی	عناصر پردازشی در هر ستون	ضرب‌کننده‌های عنصر پردازشی	درصد لایه‌های نزدیک به پیکربندی
۱	۱۶	۱	۲۵۶	۵۰
۲	۳۲	۲	۱۲۸	۲۱/۴۲
۳	۶۴	۴	۶۴	۱۴/۲۸
۴	۱۲۸	۸	۳۲	۳/۵۷
۵	۲۵۶	۱۶	۱۶	۱۰/۷۱
۶	۵۱۲	۳۲	۸	۰
۷	۱۰۲۴	۶۴	۴	۰
۸	۲۰۴۸	۱۲۸	۲	۰

با ضرب‌کننده‌های کمتر از ١٦ کارآمد نیستند و تنها حدود ۳/۵۷٪ از لایه‌ها تمایل به استفاده از عناصر پردازشی با ۳۲ ورودی دارند. به همین دلیل در طراحی درخت تجمیعی بازپیکربند با ابعاد گفته‌شده، برای گره‌های سطح یک واتسهیم‌گر و راه مستقیم به خروجی در نظر گرفته نشده تا از سربار سخت‌افزاری کاسته شود.

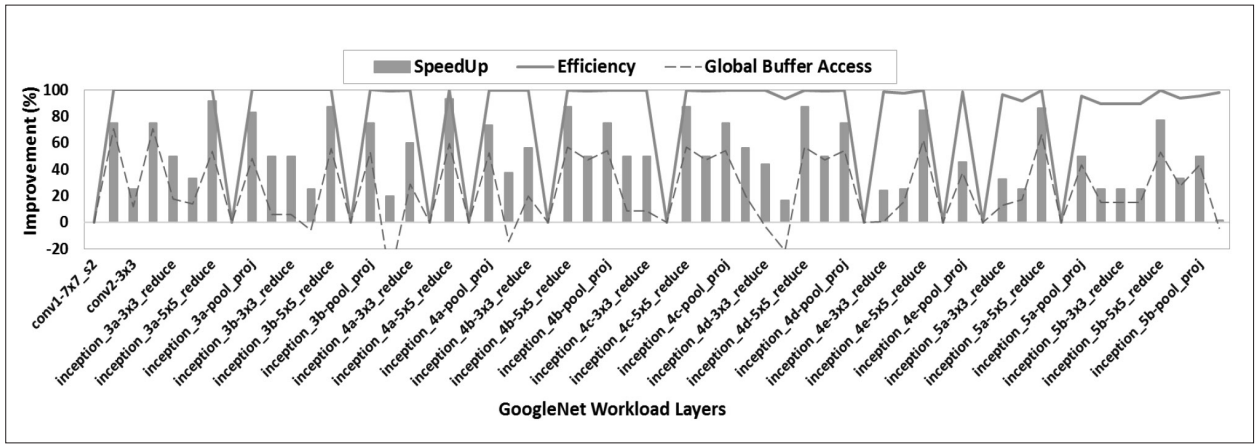
برای مثال هفتمین لایه از مدل گوگل‌نت دارای ١٦ پالایهٔ وزنی و ۱۹۲ کانال است. طبق فرمول (۱) اگر با ابعاد ساختار پایه آن را پردازش کنیم به ۱۲ چرخه نیاز داریم. چون هر ردیف مسئول اجرای یک پالایه است، پس از مجموع ١٦ کاشی، فقط یکی از کاشی‌ها با ١٦ ردیف به کار گرفته می‌شود و برای اجرای ۱۹۲ کانال توسط یک ردیف، به ۱۲ تکرار نیاز است. یعنی شتاب‌دهنده ۱۲ بار اجرا می‌شود در حالی که در هر تکرار ۱۵ کاشی بیکار است.

ساختار پیشنهادی برای این لایه، عناصر پردازشی با ٢٥٦ ضرب‌کننده را ترجیح می‌دهد. پس درخت تجمیعی بازپیکربند را طوری برنامه‌ریزی می‌کند تا خروجی عناصر پردازشی اولیه را به سمت ریشه هدایت کند. به این ترتیب هر کاشی طوری پیکربندی می‌شود که گویی یک ردیف حاوی ١٦ تا عنصر پردازشی با ٢٥٦ ضرب‌کننده دارد. هر ردیف، مسئول اجرای ۱ پالایه است. با پیکربندی جدید، تمام کاشی‌ها به کار گرفته می‌شوند (زیرا برای ١٦ پالایه به ١٦ ردیف نیاز است و در پیکربندی جدید هر کاشی حاوی یک ردیف است.). درنتیجه تنها در یک چرخه کل لایه محاسبه می‌گردد. همان‌طور که در شکل (٤) مشاهده می‌شود معماری پیشنهادی زمان محاسبات این لایه را ۹۱٪ کاهش داده است. تناسب اندازه کانال‌ها با تعداد ضرب‌کننده‌های هر عنصر پردازشی باعث کاهش دسترسی به حافظه داخلی نیز می‌شود. در این لایه میزان دسترسی به میانگیر برای خواندن وزن‌ها ٥٤٪ کاهش داشته است.

نتایج شبیه‌سازی برای مدل‌های گوناگون نشان می‌دهد که لایه‌هایی با تعداد کانال بیشتر، بهبود بهتری دارند؛ زیرا در این لایه‌ها تقاضا برای چسباندن عناصر پردازشی به یکدیگر و تولید عنصر پردازشی بزرگ‌تر بیشتر است. برای آن که بارگذاری وزن مربوط به لایه بعدی از حافظه خارجی با عملیات لایه فعلی همپوشانی داشته باشد از میانگیر دوتایی[۲۹] استفاده شده است.

به ازای هر کاشی یک میانگیر وزن و یک میانگیر خروجی وجود دارد. یک میانگیر داده ورودی نیز به تمام کاشی‌ها مشترکا متصل است. میانگیر داده ورودی در هر چرخه تعدادی کانال ورودی یکسان را برای تمام کاشی‌ها فراهم می‌کند. در حالی که در هر چرخه بر روی ردیف‌های شتاب‌دهنده پالایه‌های وزنی متفاوتی اجرا می‌شوند. هدف اصلی این معماری به حداکثر رساندن میزان بهره‌وری از منابع محاسباتی روی تراشه است. به‌طور میانگین ٦٠٪ بهبود استفاده از منابع در اجرای مدل گوگل‌نت حاصل شده

29- Double buffering



شکل ۴: بهبود سرعت محاسبات، بهره‌وری و دسترسی به میانگیر در لایه‌های مدل گوگل‌نت توسط معماری پیشنهادی نسبت به دادیانائو [۱۶]

است. با رسیدن به این هدف در کنار افزایش سرعت، میزان دسترسی به حافظه نیز کاسته می‌شود. معماری پیشنهادی ۲٤٪ از تعداد دفعات دسترسی به حافظه نسبت به معماری شیپ‌شیفتر [٦] کاسته است. سربار سخت‌افزاری که هر درخت تجمیعی بازپیکربند به سیستم اعمال می‌کند از یک عنصر پردازشی کوچک‌تر است. ساختار پیشنهادی می‌تواند بر روی اکثر شتاب‌دهنده‌های آرایه‌ای پیاده‌سازی شود و در کنار باقی روش‌های افزایش کارایی به کار گرفته شود.

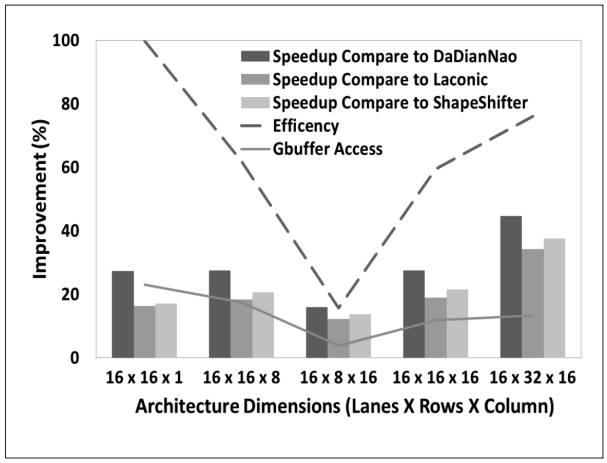
۲-۱- نتایج شبیه‌سازی‌ها

زمان انجام محاسبات، میزان استفاده از منابع و تعداد دفعات دسترسی به حافظه داخلی به‌وسیله یک شبیه‌ساز که بر اساس محاسبه چرخه‌ها نوشته شده است، انجام شد. مبنای این شبیه‌ساز در آزمایشگاه تورنتو تولید شده و مورد استفاده قرار می‌گیرد [۲٤]. مدل‌هایی که برای شبیه‌سازی استفاده شده‌اند چهار مدل یادگیری ماشین به نام‌های گوگل‌نت [۱۵]، [۲۳] VGG، رِزنت [۲۵] و شافِلنت [۲٦] هستند. مجموعه داده استفاده شده در ارزیابی‌ها ImageNet است. آزمایش‌ها مرحله بعد از آموزش را هدف قرار داده‌اند یعنی از ابتدای کار وزن‌ها آماده هستند.

شکل (۵) میزان بهبود زمان اجرای محاسبات، استفاده از منابع موجود و تعداد دفعات دسترسی به میانگیر جهت خواندن را نشان می‌دهد. در این شکل معماری پیشنهادی با ابعاد مختلف شبیه‌سازی شده و با سه معماری دیگر با همان نسبت منابع محاسباتی مقایسه شده است. همان‌طور که در شکل مشاهده می‌شود هر چه تعداد ردیف‌ها یا عناصر پردازشی داخل هر ستون بیشتر باشد، شاهد بهبود بیشتری هستیم. این به آن دلیل است که با افزایش منابعِ قابل بازپیکربند، درخت تجمیعی بازپیکربند بزرگ‌تری خواهیم داشت؛ که به این ترتیب امکان انعطاف و تطبیق‌پذیری بیشتری برای معماری فراهم می‌شود.

هر چه تعداد ورودی‌های یک عنصر پردازشی تطابق بیشتری با کانال‌های یک پالایه داشته باشد، یعنی در هر بار دسترسی بتواند تعداد کانال بیشتری را به‌صورت موازی بخواند، تعداد تکرار لازم برای خواندن تمام پالایه کمتر می‌شود. پس ساختار پیشنهادی با نزدیک کردن ابعاد هر عنصر پردازشی به تعداد کانال‌های لایه منجر به کاهش دسترسی مکرر به حافظه داخلی یا میانگیرها می‌شود. در این معماری برای شتاب‌دهنده با ابعاد ۱٦ × ۱٦ × ۱٦ دسترسی به میانگیر برای خواندن به‌طور میانگین ۱۵٪ نسبت به دادیانائو و لاکنیک کم‌تر شده و نسبت به شیپ‌شیفتر هم ۲٤٪ بهبود داشته است. سرعت انجام محاسبات نسبت به دادیانائو ۲۸٪ نسبت به لاکنیک ۱۹٪ و نسبت به شیپ‌شیفتر ۲۲٪ بیشتر شده است.

با محاسبه مساحت با کتابخانه ٤۵ نانومتر Nangate سربار سخت‌افزاری حدود ۱/۳٪ به‌دست آمده است. در



شکل ۵: افزایش سرعت، بهبود بهره‌وری و دسترسی به میانگیر هنگام اجرای مدل گوگل‌نت با معماری پیشنهادی در مقایسه با سه معماری دادیانائو [۱۶]، لاکنیک [۴] و شیپ‌شیفتر [۶] در ابعاد مختلف

ایـــن معماری از ضرب‌کننده‌های ۱۶ بیتی و جمع‌کننده‌های ۳۲ بیتی استفاده شده است.

اگر از روش‌هایی استفاده شود که با هرس‌شدن شبکه، در ابعـــاد لایه تغییراتی ایجاد شـــود، معماری پیشـــنهادی می‌تواند از ویژگی‌های مثبت شبکه‌های تُنُک به نحو بهتری نســـبت به معماری‌های پایه استفاده کند. به این ترتیب که بتواند سازگاری بیشتر با ابعاد جدید ایجاد کند. در کارهای بعدی بر روی این قابلیت تمرکز خواهد شد.

۳- نتیجه‌گیری

رشد روزافزون استفاده از شبکه‌های عصبی عمیق با کاربردهای متفاوت خصوصاً در پـــردازش تصویر، نیاز به وجود بســـتر کارا با سربار ســـخت‌افزاری کم را برای پیاده‌ســـازی آن‌ها را پدیدار کرده اســـت. محاسبات زیاد، حجـــم انتقال داده بالا و دسترســـی فراوان بـــه حافظه از ویژگی‌های مهم این شبکه‌هاست. با توجه به محدودیت‌های ســـخت‌افزاری، از منابع موجود باید بارها و بارها استفاده شود تا محاســـبات یک مدل پایان پذیرد. ساختار آرایه‌ای تکرارپذیـــر به‌عنوان یـــک معماری کارا برای شـــبکه‌های عصبی شناخته شده است. یکی از مشکلاتی که در اجرای شبکه عصبی وجود دارد، مسئله بی‌استفاده ماندن تعدادی از منابع در هر تکرار است که به دلیل اختلاف ابعاد لایه‌ها نسبت به بستر طراحی شـــده، پدیدار می‌گردد. این مسئله مانع رســـیدن به حداکثر کارایی و استفاده از تمام ظرفیت ســـخت‌افزار می‌شـــود. معماری پیشـــنهادی با استفاده از قابلیت بازپیکربندی، اندازه ابعاد شـــتاب‌دهنده را برای هر لایه طوری تغییر می‌دهد که تا جای ممکن نزدیک یا مطابق با اندازه ابعاد هر لایه شود. به این ترتیب با مصرف بهینه از منابع، سرعت انجام محاسبات مدل رزنت ۲۹/۲۸٪ افزایش یافته و میزان دسترسی به میانگیر جهت خواندن از آن ۲۲٪ کاهش یافته است. در کنار تمام این مزایا از دقت محاسبات کاسته نشده و سربار مساحت قابل چشم‌پوشی است.

مراجع

[1] Chen, Kun-Chih, et al. “NoC-based DNN accelerator: A future design paradigm.” Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip. 2019.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. “Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems 25 (2012): 1097-1105.

[3] Qin, Eric, et al. “Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training.” 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2020.

[4] Sharify, Sayeh, et al. “Laconic deep learning inference acceleration.” 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2019.

[5] Liu, Bosheng, et al. “Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators.” Proceedings of the 24th Asia and South Pacific Design Automation Conference. 2019.

[6] Chen, Shang-Tse, et al. “Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector.” Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, 2018.

[7] Jouppi, Norman P., et al. “In-datacenter performance analysis of a tensor processing unit.” Proceedings of the 44th Annual International Symposium on Computer Architecture. 2017.

[8] Chen, Yu-Hsin, et al. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks.” IEEE journal of solid-state circuits 52.1 (2016): 127-138.

[9] Kwon, Hyoukjun, Ananda Samajdar, and Tushar Krishna. “Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects.” ACM SIGPLAN Notices 53.2 (2018): 461-475.

[10] Du, Zidong, et al. “ShiDianNao: Shifting vision processing closer to the sensor.” Proceedings of the 42nd Annual International Symposium on Computer Architecture. 2015.

[19] Lee, Ching-En, et al. “Stitch-x: An accelerator architecture for exploiting unstructured sparsity in deep neural networks.” SysML Conference. 2018.
[20] Liu, Zhi-Gang, Paul N. Whatmough, and Matthew Mattina. “Systolic Tensor Array: An Efficient Structured-Sparse GEMM Accelerator for Mobile CNN Inference.” IEEE Computer Architecture Letters 19.1 (2020): 34-37.
[21] Shen, Yongming, Michael Ferdman, and Peter Milder. “Maximizing CNN accelerator efficiency through resource partitioning.” 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2017.
[22] Moons, Bert, and Marian Verhelst. “An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS.” IEEE Journal of solid-state Circuits 52.4 (2016): 903-914.
[23] Simonyan, Karen, and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition.” arXiv preprint arXiv:1409.1556 (2014).
[24] https://github.com/isakedo/DNNsim
[25] He, Kaiming, et al. “Deep residual learning for image recognition.” Proceedings of the IEEE conference on comput-er vision and pattern recognition. 2016.
[26] Ma, Ningning, et al. “Shufflenet v2: Practical guidelines for efficient cnn architecture design.” Proceedings of the European conference on computer vision (ECCV). 2018.
[11] Dave, Shail, et al. “Hardware Acceleration of Sparse and Irregular Tensor Computations of ML Models: A Survey and Insights.” arXiv preprint arXiv:2007.00864 (2020).
[12] Parashar, Angshuman, et al. “Scnn: An accelerator for compressed-sparse convolutional neural networks.” ACM SIGARCH Computer Architecture News 45.2 (2017): 27-40.
[13] Alwani, Manoj, et al. “Fused-layer CNN accelerators.” 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2016.
[14] Judd, Patrick, et al. “Stripes: Bit-serial deep neural network computing.” 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2016.
[15] Szegedy, Christian, et al. “Going deeper with convolutions.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
[16] Kwon, Hyoukjun, Ananda Samajdar, and Tushar Krishna. “A Communication-centric approach for designing flexible DNN accelerators.” IEEE Micro 38.6 (2018): 25-35.
[17] Albericio, Jorge, et al. “Bit-pragmatic deep neural network computing.” Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. 2017.
[18] Zhang, Jie-Fang, et al. “SNAP: A 1.67—21.55 TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16nm CMOS.” 2019 Symposium on VLSI Circuits. IEEE, 2019.