In Proceedings of the 4th CSI Computer Conference (CSICC'98), Jan 1999

سَدِل: از توصیف آرایه‌های تپنده تا تولید توصیف سخت‌افزاری

محمّد قدسی قاسم جابری‌پور رامتین خسروی

آزمایشگاه پردازش موازی
دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف
{ghodsi, jaberigh, raamtin}@ce.sharif.ac.ir

چکیده

آرایه‌های تپنده مدلی ساده اما کارآمد برای پردازش موازی به شمار می‌آیند. به منظور افزایش قابلیت استفاده از آنها به ابزارهایی برای توصیف، شبیه‌سازی، تحلیل، ارزیابی، وارسی و پیاده‌سازی نیاز است. لازمه‌ی ایجاد چنین ابزاری در نخستین گام یک زبان برای توصیف آرایه‌های تپنده است. زبان سَدِل [1] برای چنین کاربردی طراحی و ارائه شده است. به منظور استفاده از توصیف ارائه شده در سَدِل در ساخت آرایه‌ی تپنده، می‌توان توصیف مذکور را به زبان‌های توصیف سخت‌افزاری ترجمه کرد که به منظور برنامه‌ریزی تراشه‌های اف‌پی جی‌ای طراحی شده‌اند. در حال حاضر مترجم سَدِل توصیف آرایه‌ی تپنده را به زبان توصیف سخت‌افزاری ای‌چ‌دی‌ال ترجمه می‌کند. در این مقاله به معرفی زبان سَدِل و فرایند طراحی آرایه‌ی تپنده توسط ابزارهای طراحی شده می‌پردازیم و خصوصیات این ابزارها را در قالب مثال‌هایی بررسی می‌کنیم.

کلمات کلیدی : آرایه‌ی تپنده، زبان توصیف سخت‌افزار، پردازش موازی.

۱ مقدمه

راه حل‌های ساده و کارایی که در پی ارائه‌ی مدل آرایه‌های تپنده (systolic arrays) توسط کونگ [۱۲] برای حل مسائل پیچیده به دست آمد، توجه زیادی را به خود جلب کرد. واژه‌ی تپنده در قیاس با ضربان قلب در سیستم گردش خون بدن انسان به کار برده شده است. یک آرایه‌ی تپنده یک ساختار توری مانند از عناصر پردازشی است که داده‌ها (مقادیر ورودی به همراه نتایج میانی) در آن حرکت می‌کنند. این حرکت در جهات مختلف و احتمالاً با سرعت‌های متفاوت صورت می‌گیرد. از خصوصیات آرایه‌های تپنده نرخ بالای ورودی و خروجی است و به همین خاطر برای کاربردهای با حجم موازات بالا مناسب هستند.

اهمیت آرایه‌های تپنده را می‌توان از چند جهت بررسی کرد [۱۱]:

کاربردهای علمی با محاسبات حجیم : ماهیت تکراری و ضربانی آرایه‌های تپنده موجب می‌شود که الگوریتم‌های موازی برای حل مسائل علمی با کارایی زیاد پیاده‌سازی شوند. این کاربرد به ویژه در محاسبات ماتریسی قابل توجه است.

پردازش همروندی : آرایه‌های تپنده جمع کننده‌ی برخی از تلاش‌های پیشین برای افزودن همروندی به معماری فون‌نیومن هستند از جمله استفاده از کمک‌پردازنده‌ها، واحدهای پردازش متعدد، خط لوله‌ی داده‌ای و چندپردازنده‌های همگن.

پیشرفت‌های فن آوری : پیشرفت‌های اخیر در فن‌آوری وی‌ال‌اس‌آی مکمل قابلیت‌های آرایه‌های تپنده در حل مسائل شده‌اند. از جمله‌ی این پیشرفت‌ها می‌توان به وجود گیت‌های سریع‌تر و کوچک‌تر، بالا رفتن چگالی گیت‌ها، اقتصادی‌تر شدن فرایند طراحی و ساخت و در نهایت به میدان آمدن فن‌آوری اف‌پی جی‌ای به عنوان معماری‌های قابل پیکربندی مجدداست.

[1] SADL: Systolic Array Description Language

تراشه‌های افپی جی‌ای بستر مناسبی برای پیاده‌سازی الگوریتم‌های موازی برای آرایه‌های تپنده هستند. معماری توری‌مانند این تراشه‌ها، همچنین برنامه‌پذیر بودن آن‌ها امکان نمونه‌سازی سریع آرایه‌های تپنده را به نحو مناسبی ایجاد می‌کنند. این امر در طراحی و ساخت اقتصادی آرایه‌ها اهمیت زیادی دارد. شواهد را می‌توان در حل مسائلی از جمله حل مسأله‌ی NP-complete صدق‌پذیری (satisfiability) [۱۸]، تراز دنباله‌ها [۸] یافت.

فرایند ساخت یک مدار افپی جی‌ای برای یک الگوریتم تپنده، از فعالیت سطح بالای طراحی الگوریتم تپنده تا فعالیت سطح پایین برنامه‌ریزی تراشه‌ی افپی جی‌ای را شامل می‌شود. این فرایند به دلیل بازه‌ی متنوع فعالیت‌هایش و خطاپذیربودن آن‌ها، ایجاد ابزارهای مناسب طراحی را اجتناب ناپذیر کرده است. ابزارهای زیادی برای تبدیل یک توصیف سطح بالا از سخت‌افزار به مدار افپی جی‌ای ارائه شده است که از جمله‌ی آن‌ها می‌توان به [۱۵] و [۱۴] و [۴] اشاره کرد. در این ابزارها، محیطی برای طراحی سطح بالای واحدهای سخت‌افزاری ارائه شده است. قابلیت پیکربندی مجدد معماری افپی جی‌ای امکان طراحی انعطاف‌پذیر این واحدها را در اختیار طراح قرار می‌دهد.

در مقابل، ابزارهایی برای طراحی سطح بالای آرایه‌های تپنده ارائه شده‌اند مانند ADVIS [۱۶]، Decomposer [۱۰] و ARREST [۵]. برخی از ابزارها مانند S4CAD [۱۷] امکان تولید تمام انواع ممکن آرایه‌های تپنده برای یک توصیف سطح بالا از الگوریتم که در قالب معادلات بازگشتی بیان می‌شود را دارند. در این میان کارهایی مانند زبان CDL برای پردازش سلولی [۹] و [۲] نیز وجود دارند که زبان‌هایی برای توصیف آرایه‌های تپنده پیشنهاد کرده‌اند.

در این مقاله ابزاری برای طراحی سطح بالای آرایه‌های تپنده ارائه شده است. قسمت اصلی این ابزار که در زمان تدوین این مقاله پیاده‌سازی شده است، توصیف آرایه‌ی تپنده را در قالب زبان سَدِل می‌پذیرد و آن را به زبان توصیف سخت‌افزاری اِی‌اچ‌دی‌ال ترجمه می‌کند. در حقیقت ابزار ارائه شده، مراحل اولیه‌ی فرایند ساخت یک آرایه‌ی تپنده را از طراحی الگوریتم تا تولید یک کد میانی برای تبدیل به سخت افزار پوشش می‌دهد. این زبان دارای ویژگی‌هایی است که آن را از کارهای مشابه مانند [۹] و [۲] متمایز می‌کند از جمله بیان سراسری رفتار پردازنده‌ها، مکانیزم طراحی پیمانه‌ای و امکان تعریف هم‌بندی‌های (topology) پیچیده متشکل از پردازنده‌هایی با رفتارهای متفاوت.

در بخش ۲ به بررسی فرایند طراحی آرایه‌ی تپنده به کمک ابزار حاضر می‌پردازیم. بخش ۳ زبان سَدِل را معرفی می‌کند و به توصیف یک آرایه‌ی تپنده‌ی نمونه طبق این زبان می‌پردازد. بخش ۴ کلیات پیاده‌سازی ابزار را مرور می‌کند و در بخش ۵ جمع‌بندی و دورنمای فعالیت‌های آینده ارائه خواهد شد.

۲ آرایه‌های تپنده

فرض ما در مورد ساختار یک آرایه‌ی تپنده این است که این آرایه زیرمجموعه‌ای از یک توری از پردازنده‌ها است. هر یک از پردازنده‌های آرایه‌ی تپنده دارای چهار ورودی و چهار خروجی است که هر زوج ورودی و خروجی در یکی از جهات چپ، راست، بالا و پایین واقع شده است. طبق این مدل هر پردازنده می‌تواند از چهار پردازنده‌ی اطراف خود ورودی دریافت و به آن‌ها خروجی ارسال کند (پردازنده‌های مرزی به جای ارتباط با پردازنده‌های غایب با دنیای خارج تبادل داده می‌کنند.) این ورودی‌ها و خروجی‌ها در زبان از طریق کلمات کلیدی قابل دسترسی هستند.

هر پردازنده دارای مقدار محدود و مشخصی از حافظه‌ی محلی است که از توصیف ایستای الگوریتم قابل تعیین است. حافظه‌ی سراسری وجود ندارد و هر پردازنده فقط به حافظه‌ی محلی خود دسترسی دارد. علاوه بر حافظه‌ی محلی هر پردازنده دو اطلاع دیگر نیز ذخیره می‌کند:

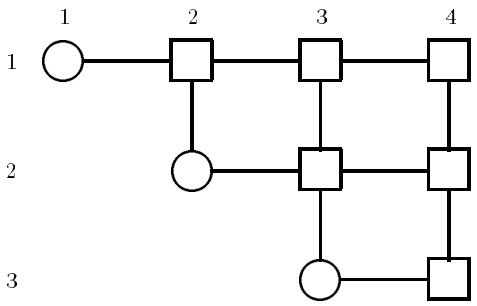
محل پردازنده: هر پردازنده از طریق متغیرهای محلی خاصی، موقعیت مکانی خود را در آرایه‌ی تپنده می‌داند. این اطلاع از طریق مکانیزم‌های زبانی به شکل محدودی در اختیار کاربر قرار می‌گیرد.

گام زمانی الگوریتم: هر پردازنده متغیری برای ذخیره‌ی گام زمانی الگوریتم دارد. از آنجا که ساعت تمام پردازنده‌ها همگام است، در هر تپش ساعت، تمام پردازنده‌ها یکی به مقدار گام زمانی خود اضافه می‌کنند.

علاوه بر حافظه‌ی محلی، هر پردازنده محاسبات خود را از طریق منطق ترکیبی روی مقادیر حافظه‌ی محلی، موقعیت مکانی و زمانی و ورودی‌ها انجام می‌دهد. تمام پردازنده‌ها دارای ساعت همگام هستند که ورودی کل آرایه محسوب می‌شود. فرض می‌شود بازه‌ی زمانی آن طوری انتخاب می‌شود که کندترین مدار پردازنده‌ها بتواند پردازش خود را بین دو تپش آن انجام دهد.

۳ زبان سدل

در این بخش به بررسی ساختارهای زبان سَدِل می‌پردازیم. این بررسی به صورت کلی انجام می‌شود. مثال‌های این بخش، نحوه‌ی استفاده از این ساختارها را نشان می‌دهند. متن کامل برنامه‌ها به همراه مثال‌های دیگر و همچنین دستورزبان سَدِل به طور کامل در [۳] آمده است.



شکل ۱: یک هم‌بندی نمونه برای حل دستگاه معادلات به روش حذف گاوسی

۱.۳ عناصر بنیادین زبان

یک برنامه‌به زبان سَدِل از سه قسمت اصلی تشکیل شده است:

اعلان انواع پردازنده‌ها: این قسمت با کلیدواژه‌ی `nodetype` شروع می‌شود. هر نوع پردازنده نام مختص خود را داشته، از قسمت‌های زیر تشکیل می‌شود:

اعلان متغیرها: در این قسمت که با کلیدواژه‌ی `var` مشخص می‌شود، حافظه‌های محلی هر پردازنده معرفی می‌شوند.

اعلان رخدادها: در این قسمت که با کلیدواژه‌ی `event` مشخص می‌شود، بروز وضعیت‌هایی از هر پردازنده که موجب بروز رفتار خاصی از جانب آن پردازنده می‌شوند، تعریف می‌شوند.

تعریف ماکروها: در این قسمت که با کلیدواژه‌ی `macro` مشخص می‌شود، دنباله‌هایی از دستورات که در تعریف بدنه‌ی متدها استفاده می‌شوند، تعریف می‌شوند. در حقیقت ماکروها نمونه‌ی ساده‌ای از مکانیزم انتزاع روی دستورات هستند که از تکرار دنباله‌های پرکاربرد جلوگیری می‌کنند.

تعریف متدها: این قسمت با کلیدواژه‌ی `method` شروع می‌شود. رفتارهای گوناگونی که هر نوع پردازنده در موقعیت‌های مکانی و زمانی خاص از خود بروز می‌دهد در قالب متدها توصیف می‌شوند. در هر متد از دنباله‌ای از دستورات انتساب و دستورات شرطی و همچنین بسط دادن ماکروها تشکیل شده است. در بدنه‌ی یک متد می‌توان به حافظه‌ی محلی پردازنده و ورودی‌ها و خروجی‌های یک پردازنده ارجاع داد.

تعریف هم‌بندی آرایه‌ی تپنده: این قسمت با کلیدواژه‌ی `network` شروع می‌شود. سپس نام آرایه‌ی تپنده و ابعاد آن اعلان می‌شوند. بعد از آن کل آرایه در صورت لزوم به چند جزء مجزا تقسیم می‌شود که هریک در قالب یک توری یا جزء تعریف شده‌ای از آن تعریف و نحوه‌ی اتصال این قسمت‌ها با هم مشخص می‌شود. بعضی از اجزاء توری در زبان از پیش تعریف شده‌اند که در تعریف هم‌بندی مورد استفاده قرار می‌گیرند.

به عنوان نمونه، آرایه‌ی که برای حل دستگاه معادلات به روش حذف گاوسی‌استفاده می‌شود (شکل ۱)، توصیفی مانند این دارد [۱۳]:

```
network
  Gauss [3,4] = tddiag[3] of Circle @ (1,1);
                upperright[3] of Square @ (1,2)
```

که در آن `Gauss` نام هم‌بندی و `Circle` و `Square` نام دو نوع پردازنده (nodetype) هستند. `tddiag` و `upperright` دو نمونه از اجزاء از پیش تعریف شده‌ی توری هستند که به ترتیب قطر اصلی و مثلث بالا—راست از توری را مشخص می‌کنند.

فعال‌سازی متدها (method enabling): این قسمت با کلیدواژه‌ی `enable` شروع می‌شود. هر جمله در این قسمت، موقعیت زمانی و مکانی خاصی را تعریف و یک یا چند متد از هر پردازنده که در آن موقعیت زمانی و مکانی قرار گیرد را فعال می‌کند. از مجموعه‌گره‌ها و مجموعه‌زمان‌ها که پیش‌تر در برنامه اعلان شده‌اند، در این قسمت استفاده می‌شود. این مجموعه‌ها به ترتیب با کلیدواژه‌های `nodeset` و `timeset` اعلان می‌شوند.

۲.۳ پیمانه‌های سدل و ترکیب آن‌ها

اگر در یک برنامه‌ی سَدِل فقط یک نوع پردازنده تعریف شده باشد، به هم‌بندی توصیف‌شده در این برنامه، هم‌بندی همگن می‌گویند، و برنامه‌های دیگر سَدِل می‌توانند این برنامه را به عنوان یک پیمانه در خود استفاده کنند. برای استفاده از یک پیمانه، برنامه‌ی استفاده‌کننده نام آن پیمانه را به عنوان یک متد وارده در قسمت متدهای خود با استفاده از کلیدواژه‌ی `import` اعلان می‌کند. مثالی از ترکیب پیمانه‌ها در برنامه‌ی بخش ۲.۳.۳ آمده است.

۳.۳ دو نمونه از توصیف آرایه‌های تپنده

در اینجادو آرایه‌ی تپنده‌ی نمونه برای مرتب کردن دنباله‌ای از اعداد را به زبان سَدِل توصیف می‌کنیم. مثال اول، الگوریتم مرتب‌سازی زوج–فرد و مثال دوم الگوریتم مرتب‌سازی مارپیچی است که در توصیف آن از توصیف الگوریتم اول استفاده می‌شود. این الگوریتم‌ها از [۱۳] اقتباس شده‌اند.

۱.۳.۳ مرتب‌سازی زوج–فرد

این الگوریتم، که روی برداری از پردازنده‌های ساده عمل می‌کند، قادر است n عدد را در n مرحله مرتب کند. در این الگوریتم، پردازنده‌ها به دو صورت دسته‌بندی شده‌اند:

- دسته‌بندی فرد: $\{(۱,۲),(۳,۴),\ldots\}$
- دسته‌بندی زوج: $\{(۲,۳),(۴,۵),\ldots\}$

در تپش‌های فرد (گام ۱، گام ۳، ...) جفت پردازنده‌های دسته‌بندی فرد مقادیر خودشان را با هم مقایسه و در صورت لزوم در جهت مرتب‌شدن بردار با هم جابجا می‌کنند. در تپش‌های زوج، جفت پردازنده‌های دسته‌بندی زوج، به صورت مشابه عمل می‌کنند.

قسمت‌های مهم برنامه‌ی سَدِل برای توصیف این الگوریتم در اینجا بررسی خواهند شد. در قسمت nodetype سه نوع پردازنده به نام‌های Square (پردازنده‌های داخلی)، LSquare (پردازنده‌ی مرزی سمت چپ) و RSquare (پردازنده‌ی مرزی سمت راست) تعریف شده‌اند. قسمتی از تعریف پردازنده‌ی Square به این شکل است:

```
Square =
var
  own : bit[4]
method
  ActRight =
  begin
    if lin > own then own := lin endif;
    rout := own
  end;
  ...
end
```

این نوع پردازنده، دارای یک متغیر محلی ۴ بیتی به نام own است که در هر لحظه یکی از n عدد را در خود ذخیره کرده است. به عنوان نمونه یک متد از این پردازنده به نام ActRight را آورده‌ایم. در بدنه‌ی این متد ابتدا مقدار ورودی چپ پردازنده با مقدار متغیر محلی مقایسه می‌شود و در صورت بزرگ‌تر بودن، مقدار ورودی چپ جایگزین متغیر محلی پردازنده می‌شود. بعد از آن در هر صورت، مقدار متغیر محلی روی خروجی راست پردازنده فرستاده می‌شود. این پردازنده دارای متدهای دیگری نیز هست که در اینجا ذکر نشده‌اند.

هم‌بندی پردازنده‌ها در این مثال، هم‌بندی ساده‌ی بردار است. برای تمایز قائل شدن بین پردازنده‌های مرزی و پردازنده‌های داخلی، پردازنده‌های مرزی به عنوان توری‌های با طول و عرض یک در محل خاص خودشان تعریف شده‌اند. پردازنده‌های داخلی نیز به عنوان توری یک بعدی با تعداد پردازنده‌ی $n-۲$ تعریف شده‌است (فرض می‌شود مقدارn قبلاً از طریق مکانیزم اعلان ثابت‌ها به عنوان یک مقدار ثابت تعریف شده‌است:)

```
network
  Net [1,n] :
    mesh [1,n-2] of Square @ (1,2);
    mesh [1,1] of LSquare @ (1,1);
    mesh [1,1] of RSquare @ (1,n)
```

مجموعه‌گره‌های تعریف شده برای این الگوریتم، گره‌های زوج (OddNodes) و گره‌های فرد (EvenNodes) هستند :

```
nodeset
  OddNodes = { ? isodd col };
  EvenNodes = { ? not isodd col }
```

مجموعه‌زمان‌های استفاده شده برای فعال‌سازی متدها شامل سه مجموعه‌زمان زیر هستند. گام زمانی اول به خاطر وضعیت خاص گره‌ها که در آن هنوز مقداری از اطراف دریافت نکرده‌اند متفاوت با سایر گام‌ها است و به همین منظور مجموعه‌زمان خاص برای آن تعریف شده است:

```
timeset
  FirstStep = step = 1;
  OddSteps = (step > 1) and isodd step;
  EvenSteps = not isodd step
```

در بخش فعال‌سازی متدها، رفتار پردازنده‌ها در شرایط زمانی و مکانی متفاوت مشخص می‌شود. به عنوان مثال دو نمونه از عبارات این بخش به این صورت هستند :

```
enable
  in OddNodes on OddSteps
    Square.ActRight,
    LSquare.ActRight,
    RSquare.ActRight
  else
  in EvenNodes on OddSteps
    Square.ActLeft,
    RSquare.ActLeft
```

```
  else
  ...
end
```

در عبارت اول، رفتار گره‌های فرد در گام‌های زمانی فرد (به جز گام ۱) مشخص شده‌است. در این گام‌ها هر یک از پردازنده‌ها از هر یک از سه نوع پردازنده، باید متد ActRight را انجام دهند.

همانگونه که مشخص است، در ساختار سَدِل وظایف پردازنده‌ها به طور محلی و در قالب متدها توصیف می‌شود. در مقابل، رفتار آن‌ها به طور سراسری و در بخش فعال‌سازی متدها تعریف شده است. این موضوع به طراح الگوریتم کمک می‌کند تا از روی درک سراسری از رفتار الگوریتم، بتواند رفتار پردازنده‌ها را تعریف کند بدون این که ناچار به شکستن رفتار سراسری به رفتارهای محلی باشد.

۲.۳.۳ مرتب‌سازی مارپیچی

در این قسمت به طور خلاصه به بررسی مثالی می‌پردازیم که ترکیب کردن پیمانه‌های سَدِل را به منظور ایجاد الگوریتم‌هایی از الگوریتم‌های ساده‌تر نشان می‌دهد. در مرتب‌سازی مارپیچی، یک توری از پردازنده‌ها در دست است که اعداد در پردازنده‌ها ذخیره شده‌اند. در مراحل فرد، توری به شکل مارپیچی مرتب می‌شود. به این معنی که سطرهای فرد، خود را از راست به چپ و سطرهای زوج، خود را از چپ به راست مرتب می‌کنند. در مراحل زوج، هر یک از ستون‌ها، خود را از بالا به پایین مرتب می‌کند. اگر اندازه‌ی هر ضلع از توری n باشد، پس از $n(۲\log n + ۱)$ مرحله، اعداد مرتب شده‌اند. الگوریتم استفاده شده برای مرتب کردن سطرها و ستون‌ها همان الگوریتم مرتب‌سازی زوج—فرد است. بنابراین هر یک از مراحل صحبت شده در بالا، خود n گام به طول می‌انجامد.

برای استفاده از الگوریتم مرتب‌سازی زوج—فرد که از قبل طراحی شده است، فرض می‌کنیم سه پیمانه‌ی سَدِل به نام‌های oeRowLeftSort برای مرتب‌کردن یک سطر از پردازنده‌ها از راست به چپ، oeRowRightSort برای مرتب‌کردن یک سطر از پردازنده‌ها از چپ به راست و oeColTopSort برای مرتب کردن یک ستون از پردازنده‌ها از بالا به پایین از قبل موجودند. فرض می‌شود این سه پیمانه طبق تعریف بخش‌های پیش همگن هستند و روی نوع پردازنده‌ای به نام Square عمل می‌کنند. به منظور بازبه‌کارگیری این پیمانه‌ها، یک نوع پردازنده به نام Square تعریف می‌کنیم و این سه پیمانه را به عنوان متدهایی از آن ذکر می‌کنیم :

```
Square :
var
  own : bit[4];
method
  import oeRowLeftSort(N);
  import oeRowRightSort(N);
  import oeColTopSort(N);
end
```

N ثابتی است که در قسمت‌های قبلی برنامه اعلان شده است و طول ضلع توری را مشخص می‌کند. هر یک از پیمانه‌های فوق، در خود ثابتی به نام N دارند که توسط آرگومان‌های مشخص شده هنگام import بازنویسی می‌شوند. به این ترتیب، فعال‌سازی متدها به این شکل انجام می‌شود (برای دیدن تعریف دقیق مجموعه‌زمان‌ها و مجموعه‌گره‌ها به [۳] مراجعه کنید:)

```
enable
  in all on oddphase
    square.oeColTopSort
  else
  in oddrow on evenphase
    square.oeRowLeftSort
  else
  in evenrow on evenphase
    square.oeRowRightSort
  else
  in all
    Square.Disable when Done
end
```

به این ترتیب ملاحظه می‌شود که مراحل الگوریتم هر یک به صورت یک بار فراخوانی پیمانه‌های تعریف شده انجام می‌شود.

۴ پیاده‌سازی

در حال حاضر پیاده‌سازی این زبان انجام شده است. در این پیاده‌سازی برنامه‌ی سَدِل به زبان توصیف سخت‌افزاری اِی‌اچ‌دی‌ال ترجمه می‌شود. دلیل انتخاب این زبان برای توصیف سخت‌افزار در دسترس بودن تراشه‌های اف‌پی‌جی‌ای ساخت شرکت آلتِرا و توصیه‌ی این شرکت در استفاده از زبان اِی‌اچ‌دی‌ال در بکارگیری تراشه‌های ساخت این شرکت به منظور به دست آوردن کارایی بالا است. مترجم سَدِل به اِی‌اچ‌دی‌ال در سیستم عامل ویندوز اجرا می‌شود. توصیف اِی‌اچ‌دی‌ال حاصل، صرفاً به منظور ارائه‌ی قابلیت پیاده‌سازی سخت‌افزاری توصیف سَدِل تولید می‌شود و برای پیاده‌سازی کارای الگوریتم روی تراشه‌ی اف‌پی‌جی‌ای ، نیاز به بازنگری در کد تولید شده توسط مترجم است.

در فرایند ترجمه ابتدا درخت نحوی انتزاعی برنامه ساخته می‌شود. بعد از آن با پیمایش درخت، توصیف‌های اِی‌اچ‌دی‌ال تولید می‌شوند. پیچیدگی عمده در تولید توصیف مقصد، تجزیه‌ی رفتار سراسری توصیف شده در برنامه‌ی سَدِل به رفتار محلی در توصیف اِی‌اچ‌دی‌ال است. در نهایت برای هر یک از نوع پردازنده‌ها یک زیرطرح در زبان اِی‌اچ‌دی‌ال تولید می‌شود. یک زیرطرح اصلی نیز به عنوان تجریدی از کل آرایه تولید می‌شود. زیرطرح اصلی، بسته به هم‌بندی شامل تعداد مناسبی از زیرطرح‌های پردازنده‌ها است که اتصالات لازم بین ورودی‌ها و خروجی‌های آن‌ها نیز برقرار می‌شود. مثالی از خروجی تولید شده در [۳] آمده است.

۵ نتیجه‌گیری

در جمع‌بندی می‌توان گفت زبان سَدِل از ساختارهای مناسبی برای بیان سطح بالای الگوریتم‌های مبتنی بر آرایه‌های تپنده برخوردار است. جداکردن وظایف محلی از رفتار سراسر در پردازنده‌ها کار طراح الگوریتم را ساده‌تر می‌کند. همچنین توصیف رفتار مبتنی بر موقعیت‌ها قابلیت فهم یک توصیف به زبان سَدِل را بالا می‌برد. امکان تعریف و ترکیب پیمانه‌ها در سَدِل امکان بازبه‌کارگیری توصیف‌ها را در اختیار طراح الگوریتم قرار می‌دهد. با توجه به نکات فوق کاربردهای بالقوه‌ی زبان سَدِل می‌تواند شامل موارد زیر باشد:

طراحی سخت‌افزار: با رایج شدن سخت‌افزارهای **معماری‌های قابل پیکربندی مجدد**[۲] مانند تراشه‌های اف‌پی جی‌اِی و وجود نرم‌افزارهای برنامه‌ریزی این سخت‌افزارها، امکان تبدیل خودکار توصیف آرایه‌ها در زبان سَدِل به این گونه سخت‌افزارها ممکن است. نمونه‌ای از این کار در پیاده‌سازی فعلی در تبدیل سَدِل به اِی‌اچ‌دی‌ال قابل مشاهده است.

ابزار شبیه‌سازی: از این زبان می‌توان برای تولید ابزار شبیه‌سازی آرایه‌های تپنده استفاده کرد. چنین ابزاری می‌تواند طراح را در بررسی طرح خود یاری دهد. به‌علاوه از این ابزار برای مقاصد آموزشی نیز می‌توان استفاده کرد.

ابزار وارسی: به منظور اطمینان از صحت عملکرد الگوریتم می‌توان از زبان سَدِل برای توصیف آرایه‌ی تپنده برای یک ابزار وارسی استفاده کرد. با پیشرفت‌های اخیر در روش‌های وارسی خودکار (مثلاً مبتنی بر منطق زمانی) تولید چنین ابزاری کاملاً ممکن است. نمونه‌ای از چنین کاربردی را می‌توان در استفاده از ابزار کاسپَن [۷] در وارسی توصیف‌های سخت‌افزاری اشاره‌کرد [۶].

مقاصد آموزشی: از آنجا که هیچ زبان مرجعی برای توصیف آرایه‌های تپنده وجود ندارد، سَدِل می‌تواند در متون آموزشی برای توصیف الگوریتم‌ها جایگزین جملات زبان طبیعی شود. این موضوع دقت بیان الگوریتم‌ها را بالا می‌برد و به ویژه در کنار ابزارهای فوق می‌تواند بسیار مفید باشد.

گسترش‌های ممکن برای این ابزار می‌تواند شامل موارد زیر باشد:

- بهبود ساختارهای زبان: با مطالعه‌ی نظری زبان سَدِل و تشریح صوری معنای زبان توسط روش‌های صوری متعارف، بهبود ساختارهای زبان و به دست آوردن زبانی با ساختارهای کمینه و سازگار با یکدیگر را ممکن می‌سازد.
- تولید ابزارهای شبیه‌سازی و ابزارهای وارسی که پیش از این مطرح شدند. این کار می‌تواند بر مبنای ابزار وارسی ارائه شده در [۱] که بر مبنای منطق زمانی طراحی شده‌است، صورت بگیرد.
- مطالعه‌ی بیشتر روی ساختار پیمانه و ترکیب پیمانه‌ها به منظور ایجاد امکان تولید یک کتابخانه از آرایه‌های تپنده با قابلیت ترکیب.
- ترجمه‌ی سَدِل به زبان‌های توصیف سخت‌افزاری قوی مانند وی‌اچ‌دی‌ال به منظور استفاده از امکانات این زبان‌ها برای تولید توصیف‌های کاراتر. همچنین بهبود مترجم سَدِل یا تهیه‌ی یک بهینه‌ساز برای آن به منظور به دست آوردن توصیف سخت‌افزاری بهتر و کاراتر.
- برنامه‌ریزی مستقیم سخت‌افزار: در فرایند طراحی فوق، به خاطر وجود زبان میانی مانند اِی‌اچ‌دی‌ال کیفیت طرح تولید شده از نظر کارایی نسبتاً پایین خواهد بود. با برداشتن این زبان میانی می‌توان مستقیماً سَدِل را برای برنامه‌ریزی تراشه‌های اف‌پی جی‌اِی به کاربرد و به این ترتیب نتیجه‌ای با کارایی بیشتر به دست آورد.

مراجع

[۱] ر. خسروی و ع. موقر، *اعتبارسنجی سیستم‌های همروند به کمک مدل اکتور و منطق زمانی*، سومین کنفرانس سالانه انجمن کامپیوتر ایران، دی ۱۳۷۶، دانشگاه علم و صنعت ایران.

[۲] reconfigurable

[۲] آ. سینایی خسروشاهی و م.ر. میبدی، *یک زبان سطح بالا برای برنامه‌سازی الگوریتم‌های تپنده*، سومین کنفرانس سالانه انجمن کامپیوتر ایران، دی ۱۳۷۶، دانشگاه علم و صنعت ایران.

[۳] م. قدسی، ق. جابری‌پور و ر. خسروی، *معرفی زبان سَدِل برای طراحی آرایه‌های تپنده*، گزارش فنی، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف، ۱۳۷۷.

[4] S.J.B. Acockand K.R. Diamond, Automatic Mapping of Algorithms onto Multiple FPGA-SRAM Modules, *Field-Programmable Logic and Applications'97*, LNCS Vol. 1304, Springer-Verlag, New York, 1997.

[5] W. Burleson and B. Jang, ARREST: an Interactive graphic Analysis Tool for VLSI Arrays, *Proc. Int. Conf. on Application Specific Array Processors*, 1992.

[6] K. Fisler and R. Kurshan, Verifying VHDL Designs with COSPAN, in T. Kropf (ed.), *Formal Hardware Verification*, LNCS Vol. 1287, Springer-Verlag, New York, 1997.

[7] R. Hardin, Z. Har'El and R. Kurshan, COSPAN, *Computer Aided Verification'96 Conference Proceedings*, LNCS Vol. 1102, Springer-Verlag, New York, 1996.

[8] D.T. Hoang and D.P. Lopresti, FPGA implementation of Systolic Sequence Alignment, *Field-Programmable Gate Arrays'92*, LNCS Vol. 705, Springer-Verlag, New York, 1992.

[9] C. Hochberger and R. Hoffmann, CDL - A Language for Cellular Processing, *MPCS'96: Proc. of 2nd Int. Conf. on Massively Parallel Computing Systems*, pp. 48-53, IEEE Computer Society,1996.

[10] P.P. Hon, R.M. Ownes and M.J. Irwin, DECOMPOSER: a Synthesizer for Systolic Systems, *Proc. of Int. Conf. on Systolic Arrays*, 1989.

[11] K.T. Johnson, A.R. Hurson and B.Shirazi, General-Purpose Systolic Arrays, *Computer*, Vol. 26, No. 11, Nov. 1993.

[12] H.T. Kung, Why Systolic Architectures?, *Computer*, Vol. 15, No. 1, Jan. 1982.

[13] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, San Mateo, 1992.

[14] W. Luk and S. McKeever, Pebble: A Language for Parameterised and Reconfigurable Hardware Design, *Field-Programmable Logic and Applications'98*, LNCS Vol. 1482, Springer-Verlag, New York, 1998.

[15] G. McGregor, D. Robinson and P. Lysaght, A Hardware/Software Co-design Environment for Reconfigurable Logic Systems, *Field-Programmable Logic and Applications'98*, LNCS Vol. 1482, Springer-Verlag, New York, 1998.

[16] D.I. Moldovan, ADVIS: a Software Package for the Design of Systolic Arrays, *IEEE Trans. on Computer Aided Design*, Vol. 6, No. 1, 1987.

[17] S.G. Sedukhin and I.S. Sedukhin, Systematic Approach and Software Tool for Systolic Design, *Proc. of Parallel Processing: CONPAR'94-VAPP VI*, LNCS Vol. 854, Springer-Verlag, New York, 1994.

[18] P. Zhong, M. Martonos, P. Ashar and S. Malik, Solving Boolean Satisfiability with Dynamic Hardware Configurations, *Field-Programmable Logic and Applications'98*, LNCS Vol. 1482, Springer-Verlag, New York, 1998.