نشریه علمی "پدافند الکترونیکی و سایبری"
سال هفتم، شماره ۴، زمستان ۱۳۹۸، ص ۶۵-۵۹

طراحی آرایه سیستولیکی برای اجرای الگوریتم SL0

علی ناصری[1*]، روزبه جزپیری[2]
۱- دانشیار، دانشگاه جامع امام حسین(ع) ۲- کارشناس ارشد الکترونیک، دانشگاه زنجان

(دریافت: ۹۷/۱۲/۰۵، پذیرش: ۹۸/۰۳/۲۸)

چکیده

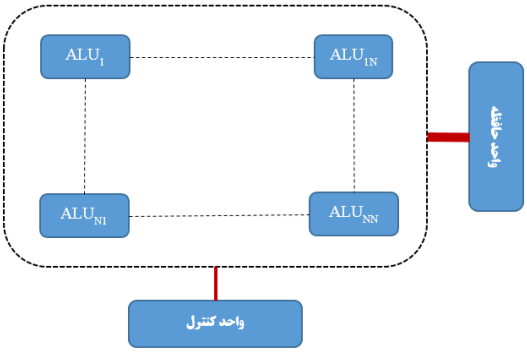
معماری سیستولیکی یکی از پرکاربردترین معماری‌های پردازش موازی به حساب می‌آید. در آرایه سیستولیکی واحدهای ALU به‌صورت آرایه کنار هم قرار می‌گیرند. این آرایه به‌صورت سنکرون عمل می‌کند به‌صورتی که با نگاشت مناسب ورودی‌ها به آن قادر است محاسبات دارای معادله بازگشتی را به‌طور موازی انجام دهد. در این مقاله آرایه سیستولیکی برای یکی از الگوریتم‌های استفاده‌شده در نمایش (تجزیه) تنک بنام الگوریتم $SL0$ طراحی و با شبیه‌سازی نرم‌افزاری مورد ارزیابی واقع گردید. نتایج حاکی از آن است که اجرای الگوریتم مذکور با تک پردازنده با فرض ۴ کلاک برای انجام هر بار معادله بازگشتی تعداد کلاکی معادل $N^3 + 9.7N^2 + 3.2N + 18$ نیاز دارد در حالی که انجام آن با آرایه سیستولیکی به‌دلیل انجام محاسبات به‌صورت موازی و پایپ لاین، کلاکی معادل $48N + 32$ لازم دارد.

کلیدواژه‌ها: آرایه سیستولیکی، پردازش موازی، الگوریتم SL0، ضرب ماتریس

۱. مقدمه

الف) آرایه سیستولیکی

پردازش موازی در سطوح مختلف مد نظر محققین حوزه پردازش بوده و خواهد بود. یکی از مهم‌ترین سطوح پردازش موازی، انجام محاسبات در سطح واحدهای محاسبات و منطق است. موازی‌سازی واحدهای ALU سرعت انجام بخش عمده‌ای از عملیات را دوچندان می‌نماید. شکل (۱) بلوک دیاگرام پردازنده موازی در سطح ALU را نشان می‌دهد.



شکل (۱): پردازنده موازی در سطح ALU [۱]

در شکل (۱)، واحد کنترل یک دستورالعمل را از حافظه فراخوانی می‌کند. سپس آن را دیکد کرده و داده‌های مورد نیاز آن را از حافظه فراخوانده و با نگاشت مناسبی به واحدهای ALU اعمال می‌نماید. این واحدهای به‌طور موازی عملیات را انجام می‌دهند. دو حالت برای عمل نمودن واحدهای ALU وجود دارد که هر کدام منشأ معماری خاصی می‌باشند. عملکرد سنکرون واحدهای ALU را معماری سیستولیک و عملکرد آسنکرون واحدهای ALU را معماری "ویو فرونت" نام نهاده‌اند.

یک آرایه سیستولیکی ساختاری توری مانند از عناصر پردازشی و تأخیر است که داده‌ها، مقادیر ورودی و نتایج میانی، در جهات مختلف حرکت می‌کنند. آرایه‌های سیستولیکی به‌دلیل بهره‌مندی از ساختار مشابه موازی و خط لوله‌ای، فعالیت هم‌زمان، نرخ بالای ورودی و خروجی و برخورداری از ساختار نسبتاً همگن به‌ویژه در پردازنده‌های غیر مرزی، از معماری‌های برگزیده برای پردازش موازی به شمار می‌روند. ساختار آرایه سیستولیکی، در کاربردهای با حجم بالای محاسبات مانند محاسبات ماتریسی، محاسبات مثلثاتی [۶-۳]، کانوولوشن[1]، همبستگی[2]، تجزیه و تحلیل دنباله DNA [۷]، رادارها و سامانه‌های مخابراتی [۹-۸]، الگوریتم‌های برنامه‌نویسی پویا[3] و مسائل ریاضی دارای معادله بازگشتی، مورداستفاده قرار می‌گیرد.

ب) روش تُنُک

دستگاه‌های معادلات خطی «فرومعین»[4]، دستگاه‌هایی هستند که تعداد مجهول‌های آن‌ها از تعداد معادله‌هایشان بیش‌تر است. این

* رایانامه نویسنده مسئول: anaseri@ihu.ac.ir

[1] Convolution
[2] Correlation
[3] Dynamic Programming
[4] Underdetermined Systems of Linear Equations (USLE)

دستگاه‌های معادلات بی‌نهایت جواب دارند. دانشمندان به دنبال یافتن تُنُک‌ترین پاسخ برای این معادلات بوده‌اند [۱۰].

پیدا کردن تُنُک‌ترین پاسخ از این‌رو اهمیت دارد که در برخی کاربردها، نظیر موارد زیر استفاده می‌گردد:

- تصویربرداری تشدید مغناطیسی[1] (MRI) [۱۱].
- خوشه‌بندی [۳].
- جداسازی کور منابع[2] [۱۳-۱۲].
- مخابرات و تخمین کانال[3] [۱۴].
- نمونه‌برداری فشرده [۳].
- رادار[4] [۱۵].
- بینایی ماشین[5] [۱۶]

فرض شود که تعداد معادله‌های یک دستگاه معادلات از تعداد مجهولات آن کم‌تر باشد. به عبارتی، در دستگاه معادلات $y = Ax$ که هدف پیدا کردن بردار x است، تعداد سطرهای ماتریس A کم‌تر از تعداد ستون‌هایش باشد. به این مسئله، یک مسئله فرومعین گفته می‌شود. در این حالت دستگاه معادلات جواب یکتایی نداشته و دارای بی‌شمار پاسخ است. تعداد ستون‌های ماتریس A که به ماتریس دیکشنری[6] معروف است، با تعداد درایه‌های بردار x (تعداد مجهولات) و تعداد سطرهای آن با تعداد درایه‌های بردار y (تعداد معادلات) برابر است. ماتریس A که تعداد سطرهایش کم‌تر از تعداد ستون‌هایش است، را ماتریس فراکامل[7] می‌نامند.

چون معادله $y = Ax$ با توجه به شرایط ذکرشده، بی‌شمار جواب دارد؛ پس باید شرایطی اعمال شود تا بتوان جواب یکتایی تحت آن شرایط برای دستگاه معادلات پیدا نمود. لذا ایده یافتن تنک‌ترین پاسخ مطرح می‌شود. تنک‌ترین پاسخ به جوابی گفته می‌شود که تعداد درایه‌های صفر آن، حداکثر تعداد ممکن باشد. "الاد" و "دونوهو" در [۱۷] ثابت کرده‌اند که این پاسخ یکتاست.

در معادله $y = Ax$ ماتریس A، ماتریس دیکشنری نام دارد [۱۸] و به‌صورت زیر است:

$$A = \underset{\in \mathbb{R}^{m\times n}}{[a_1.a_2.\dots.a_n]} \qquad (۱)$$

هر یک از ستون‌های آن را که به‌صورت $a_k \in \mathbb{R}^{m\times 1}$ هستند، یک اتم[7] از ماتریس A می‌نامند [۱۸]. در حقیقت حل این دستگاه معادلات، معادل پیدا کردن یک ترکیب خطی برای بردار $y \in \mathbb{R}^{m\times 1}$ برحسب اتم‌های ماتریس دیکشنری است. این ترکیب خطی به‌صورت زیر در نظر گرفته می‌شود:

$$y = a_1x_1 + a_2x_2 + \cdots + a_nx_n = Ax \qquad (۲)$$

$$x = [x_1.x_2.\dots.x_n]^T \in \mathbb{R}^{n\times 1} \qquad (۳)$$

درایه‌های بردار ضرایب اتم‌ها در این ترکیب خطی است. پیدا کردن تُنُک‌ترین پاسخ برای دستگاه معادلات بیان‌شده، معادل یافتن ترکیب خطی برای بردار y است که در آن کمترین تعداد اتم‌های ماتریس دیکشنری مورداستفاده قرار گرفته باشد.

برای یافتن تُنُک‌ترین پاسخ، باید تعداد درایه‌های غیر صفر بردار x که بردار ضرایب[8] نامیده می‌شود، حداقل شود. تعداد درایه‌های یک بردار، با نُرم[9] صفر آن بردار نشان داده می‌شود. لذا برای یافتن تنک‌ترین پاسخ معادله $y = Ax$ باید مسئله بهینه‌سازیِ شرطی زیر حل شود:

$$\min_x \|x\|_0 \quad subject\ to \quad y = Ax \qquad (۴)$$

عملگر $\|\cdot\|_0$ بیانگر نُرم صفر بردار است.

۲. الگوریتم $SL0$

یکی از الگوریتم‌های استفاده‌شده در نمایش (تجزیه) تنک، الگوریتم $SL0$ است. مراحل الگوریتم مذکور به شرح زیر است:

۱- مقداردهی اولیه

الف) مقدار اولیه بردار x را برابر $x = A^{\dagger}y$ قرار می‌دهیم. ($A^{\dagger}$ شبه معکوس ماتریس است.)

ب) انتخاب یک دنباله کاهش مناسب از مقادیر σ: $[\sigma_1.\sigma_2.\dots.\sigma_K]$

پ) انتخاب یک مقدار مناسب برای μ

۲- تکرار مراحل زیر به ازای $k = 1.2.\dots.K$

الف) $\sigma = \sigma_k$

ب) تکرار L مرتبه الگوریتم سریع‌ترین رشد و یافتن مقدار حداکثرکننده تابع $f_\sigma(x)$ و تصویر کردن پاسخ به‌دست‌آمده بر روی مجموعه پاسخ‌های معادله $y = Ax$ یعنی مجموعه $\chi = \{x|\ y = Ax\}$ و در حالت نویزی، مجموعه زیر است:

$$\chi = \{x|\ \|y - Ax\|_2 \le \varepsilon\}$$

ب.۱) مقداردهی اولیه $x = \hat{x}_{k-1}$

ب.۲) L مرتبه تکرار مراحل زیر به ازای $l = 1.2.\dots.L$

ب.۱.الف) $$\nabla F_\sigma(\mathrm{x}) = -\frac{1}{\sigma^2}\left[x_1e^{-\frac{x_1^2}{2\sigma^2}}.x_2e^{-\frac{x_2^2}{2\sigma^2}}.\dots.x_ne^{-\frac{x_n^2}{2\sigma^2}}\right]^T$$

[1] Magnetic Resonance Imaging
[2] Blind Source Separation (BSS)
[3] Communication and Channel Estimation
[4] Radar
[5] Machine Vision
[6] Dictionary matrix
[7] Atom
[8] Coefficients Vector
[9] Norm

ب.۱.ب) بردار x به‌روز شود:

$$x \leftarrow x + \mu\sigma^2 \nabla F_\sigma(x)$$

ب.۱.ج) بردار x بر مجموعه χ تصویر شود:

$$x \leftarrow x - A^T(AA^T)^{-1}(x - A^{\dagger}\mathrm{y})$$

ب.۳) $\hat{x}_k = x$

پاسخ نهایی برابر است با $\hat{x} = x_K$

اجرای این الگوریتم به‌دلیل وجود ضرب ماتریس‌های پیچیده اعشاری زمان‌بر است. فلذا بهره‌گیری از معماری سیستولیک طراحی‌شده، می‌تواند موجب تسریع در اجرای محاسبات الگوریتم مذکور شود.

۳. طراحی آرایه سیستولیکی ضرب ماتریس

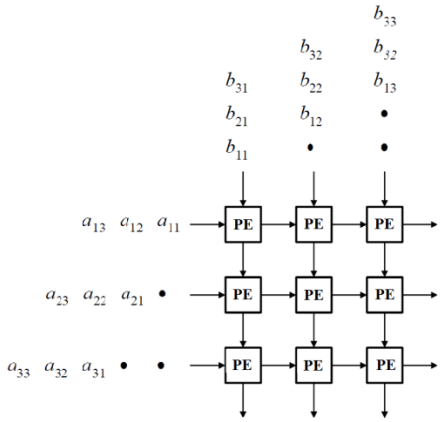
در این بخش هدف این است که یک‌ضرب کننده ماتریسی با استفاده از معماری سیستولیکی طراحی و شبیه‌سازی شود. فرض شود که A و B هرکدام ماتریس‌هایی با ابعاد $N \times N$ باشند. حاصل‌ضرب این دو ماتریس، یک ماتریس با ابعاد $N \times N$ به نام C است که به‌صورت زیر خواهد بود:

$$A_{N\times N} \times B_{N\times N} = C_{N\times N} \quad (۵)$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1N} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NN} \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{1N} \\ \vdots & \ddots & \vdots \\ c_{N1} & \cdots & c_{NN} \end{bmatrix} \quad (۶)$$

$$C_{ij} = \sum_{k=1}^{N} A_{ik} \cdot B_{kj} \quad (۷)$$

برای محاسبه هر درایه ماتریس C، باید حاصل جمع تجمعی رابطه (۷) محاسبه شود که ترکیبی از عملیات جمع و ضرب است. شکل (۲) یک آرایه سیستولیک ۳ × ۳ را نشان می‌دهد.



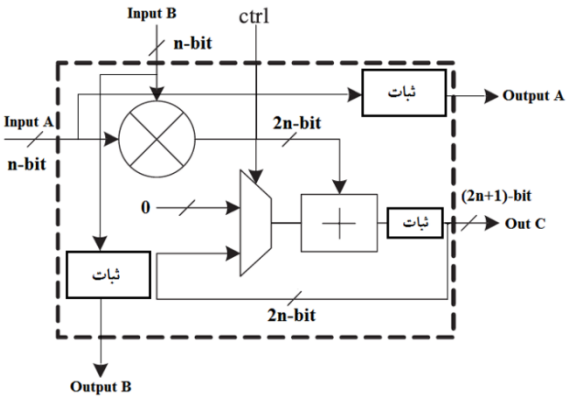
شکل (۲): آرایه سیستولیکی دوبعدی

با توجه به رابطه (۷)، هر عنصر پردازشی آرایه باید توانایی انجام دو عمل زیر را داشته باشد:

۱- محاسبه حاصل‌ضرب دو عملوند باینری ورودی (A_{ik} و B_{kj})

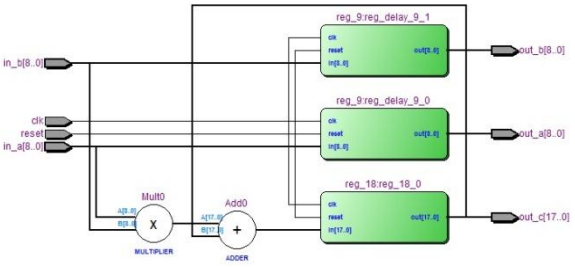
۲- محاسبه حاصل جمع تجمعی حاصل‌ضرب‌ها

بنابراین برای انجام این ضرب توسط آرایه سیستولیکی، هر عنصر پردازشی باید حاوی یک جمع‌کننده و ضرب‌کننده با تعداد بیت ورودی موردنیاز باشد. این ساختار پردازشی در شکل (۳) نشان داده شده است.



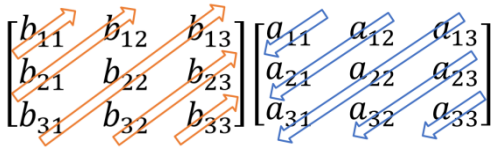
شکل (۳): یک واحد پردازشی آرایه سیستولیکی برای ضرب ماتریس

همان‌طور که در شکل (۳) مشاهده می‌شود در لبه کلاک، داده‌های ورودی (که شامل درایه‌های ماتریس A و B یعنی Input A و Input B است) با تعداد بیت‌های معین، به داخل عنصر پردازشی (PE) پمپ می‌شوند. در ابتدا عمل ضرب توسط واحد ضرب کننده روی داده‌ها انجام می‌گیرد. سپس حاصل با نتایج قبلی (جمع تجمعی حاصل‌ضرب‌ها در کلاک‌های قبل) جمع می‌شود. در لبه کلاک بعدی، داده‌های دریافت شده در کلاک قبل (Output A و Output B) از طریق مسیرهای داده پیش‌بینی‌شده، به عنصر پردازشی همسایه به‌عنوان ورودی‌های جدید ارسال می‌شوند. لازم به ذکر است برای حفظ داده‌های ورودی در هر سیکل کاری (جهت انتقال به عنصر همسایه) و همچنین ذخیره نتایج پردازش هر واحد، سه عدد ثبات در داخل هر عنصر پردازشی تعبیه شده است. شکل (۴)، PE طراحی‌شده بر اساس ساختار مذکور را نشان می‌دهد. طراحی و شبیه‌سازی با استفاده از زبان توصیف سخت‌افزار Verilog در محیط Quartus انجام شده است. لازم به ذکر است صحت عملکرد واحد طراحی‌شده با استفاده از شبیه‌ساز Modelsim نسخه Altera Edition مورد آزمون قرار گرفته است.



شکل (۴): نمونه عنصر پردازشی طراحی‌شده برای ورودی‌های ۹ بیتی

ترتیب استفاده از درایه‌های ماتریس A و B با ابعاد 3×3 در شکل (۵) آورده شده است.



شکل (۵): ترتیب انتقال درایه‌های ماتریس‌های **A** و **B** به داخل آرایه سیستولیک به‌عنوان داده‌های ورودی

برای تبیین موضوع و درک بهتر از عملکرد این ساختار، در جدول ۱ عملیات پردازشی برای ضرب دو ماتریس 3×3 در هر کلاک آورده شده است.

جدول (۱): زمان‌بندی ضرب دو ماتریس 3×3 با ساختار آرایه سیستولیکی

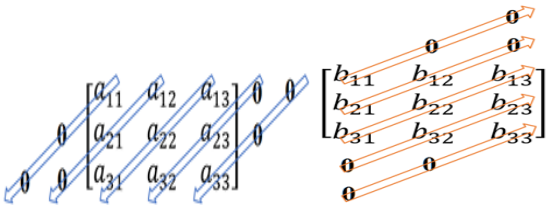
Clock / C_{ij}	۱	۲	۳	۴	۵	۶	۷
C_{11}	$a_{11}b_{11}$	$a_{12}b_{21}$	$a_{13}b_{31}$				
C_{12}		$a_{11}b_{12}$	$a_{12}b_{22}$	$a_{13}b_{32}$			
C_{21}		$a_{21}b_{11}$	$a_{22}b_{21}$	$a_{23}b_{31}$			
C_{31}			$a_{31}b_{11}$	$a_{32}b_{21}$	$a_{33}b_{31}$		
C_{13}			$a_{11}b_{13}$	$a_{12}b_{23}$	$a_{13}b_{33}$		
C_{22}			$a_{21}b_{12}$	$a_{22}b_{22}$	$a_{23}b_{32}$		
C_{23}				$a_{21}b_{13}$	$a_{22}b_{23}$	$a_{23}b_{33}$	
C_{32}				$a_{31}b_{12}$	$a_{32}b_{22}$	$a_{33}b_{32}$	
C_{33}					$a_{31}b_{13}$	$a_{32}b_{23}$	$\boldsymbol{a_{33}b_{33}}$

باید دقت شود ابعاد آرایه سیستولیکی متناظر با ابعاد ماتریس پاسخ یا همان ماتریس C خواهد بود یعنی اگر داشته باشیم:

$$A_{N\times M} \times B_{M\times K} = C_{N\times K} \qquad (۸)$$

در این صورت، ابعاد آرایه سیستولیکی $N \times K$ خواهد بود.

یک نکته قابل‌توجه دیگر این است که با توجه به چینش عناصر پردازشی و ورود داده‌ها از عناصر مرزی، در کلاک‌های ابتدایی و انتهایی برخی از واحدها داده‌ای دریافت نمی‌کنند. به عبارتی‌دیگر داده پمپ شده به داخل آرایه پس از چندین کلاک به آن‌ها می‌رسد. برای جلوگیری از بروز خطا در پاسخ نهایی، در طول این کلاک‌ها، مقدار صفر به عناصر مذکور تزریق می‌شود. شکل (۶) تزریق داده‌ها به داخل آرایه را نشان می‌دهد.



شکل (۶): ترتیب تزریق داده به داخل آرایه

برای آزمودن ساختار فوق، از اعداد تصادفی ایجادشده توسط نرم‌افزار MATLAB، استفاده شده است. اعداد تصادفی تولیدشده قبل از تزریق به داخل آرایه در واحدهای حافظه به ترتیب مشخص شده در شکل (۷ و ۸) ذخیره می‌شوند.

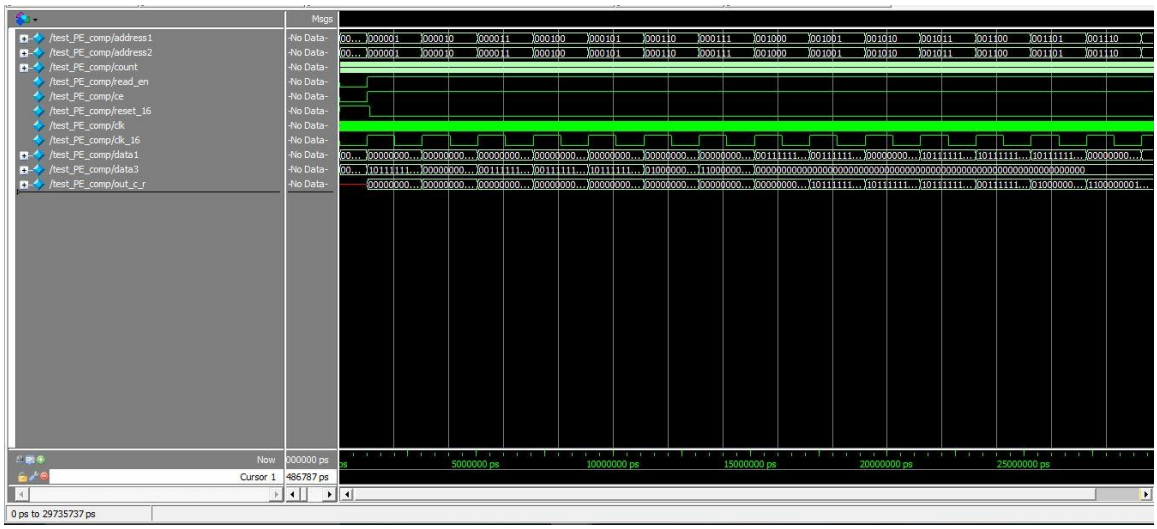
0	0	a_{11}	0
0	a_{21}	a_{12}	1
a_{31}	a_{22}	a_{13}	2
a_{32}	a_{23}	0	3
a_{33}	0	0	4

شکل (۷): ترتیب ذخیره‌سازی درایه‌های ماتریس A در داخل واحد حافظه

0	0	b_{11}	0
0	b_{12}	b_{21}	1
b_{13}	b_{22}	b_{31}	2
b_{23}	b_{32}	0	3
b_{33}	0	0	4

شکل (۸): ترتیب ذخیره‌سازی درایه‌های ماتریس B در داخل واحد حافظه

با بهره‌گیری از استاندارد IEEE 754، واحدهای ضرب‌کننده و جمع‌کننده طراحی‌شده را تغییر می‌دهیم تا امکان پردازش و انجام عملیات روی داده‌های ورودی اعشاری، توسط عناصر پردازشی سیستولیک امکان‌پذیر شود. با توجه به اینکه عمل ممیز شناور با دقت مضاعف (۶۴ بیتی) در نظر گرفته شده است؛ لذا باید ظرفیت ثبات‌ها برای ذخیره داده‌های میانی با توجه به آن انتخاب شود و در غیر این صورت پاسخ درستی حاصل نخواهد شد. آرایه سیستولیکی اعشاری پس از طراحی و اعمال الگوریتم ممیز شناور با استفاده از شبیه‌ساز Modelsim با داده‌های تصادفی آزمایش و صحت‌سنجی شد. نتایج حاصل از شبیه‌سازی در شکل (۹) مشاهده می‌شود.



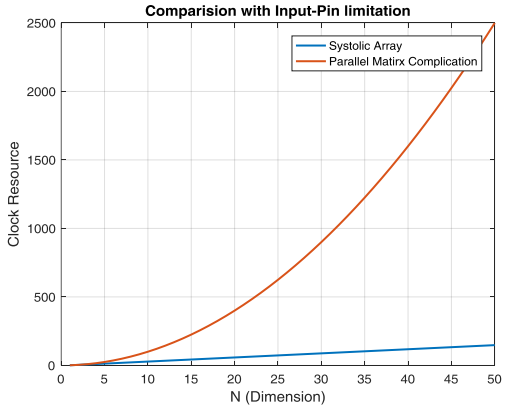
شکل (۹): نتایج حاصل از آزمون ضرب‌کننده اعشاری

۴. ارزیابی طراحی

الف) ارزیابی آرایه سیستولیکی ضرب ماتریسی

برای ضرب دو ماتریس با ابعاد $N \times N$ به‌صورت تک پردازنده‌ای سری، پیچیدگی زمانی از مرتبه N^3 خواهد بود، به‌طوری‌که N^3 عمل ضرب، $N \times 2(N-1)$ عمل جمع نیاز است. ساختار آرایه سیستولیکی طراحی‌شده تنها به $(3N-2)$ دوره عملیاتی جهت انجام پردازش مشابه نیاز دارد.

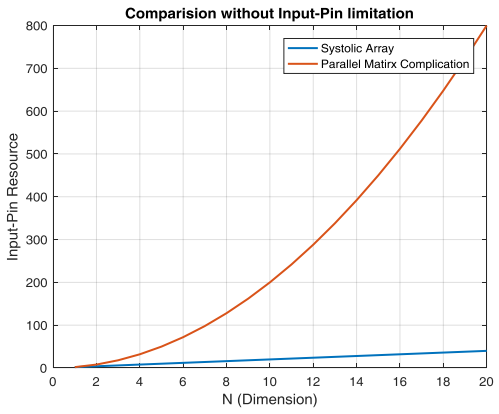
حال ضرب دو ماتریس را یک‌بار با استفاده از آرایه سیستولیکی طراحی‌شده و بار دیگر در حالت عادی (چندپردازنده بدون استفاده از آرایش آرایه‌ای و نه تک پردازنده‌ای) انجام می‌دهیم. ابعاد ماتریس‌ها را از ۱ تا ۵۰ تغییر می‌دهیم. با محدود در نظر گرفتن پایه‌های ورودی، نتیجه حاصل از دو روش ازنظر تعداد کلاک‌های لازم در شکل (۱۰) نشان داده شده است.



شکل (۱۰): مقایسه پیچیدگی زمانی در شرایطی که تعداد پایه‌های ورودی محدود باشد.

توجه شود منظور از محدود در نظر گرفتن پایه‌های ورودی این است که تعداد ورودی‌های ضرب ماتریس در حالت عادی را برابر با تعداد پایه‌های ورودی ضرب سیستولیکی قرار دهیم. برای مثال اگر ابعاد ماتریس ۱۰×۱۰ باشد، تعداد پایه‌های ورودی در آرایش سیستولیکی برابر با ۲۰ خواهد بود و کل عملیات ضرب، در ۲۸ کلاک انجام خواهد شد. اگر ورودی‌ها را به همین تعداد محدود کنیم، با آرایش معمولی (چندین واحد پردازشی که اتصال آرایه‌ای ندارند) تعداد ۱۰۰ کلاک برای انجام عمل ضرب ماتریس لازم است.

همین آزمایش را بار دیگر بدون در نظر گرفتن محدودیت برای پایه‌های ورودی دنبال می‌کنیم. حاصل در شکل (۱۱) نمایش داده شده است. مشاهده می‌شود با افزایش ابعاد ماتریس تعداد پایه‌های موردنیاز در حالت عادی با روند قابل‌توجهی (از درجه ۲) افزایش می‌یابد درصورتی‌که این تغییر برای معماری سیستولیکی به‌صورت خطی با شیب کم است.

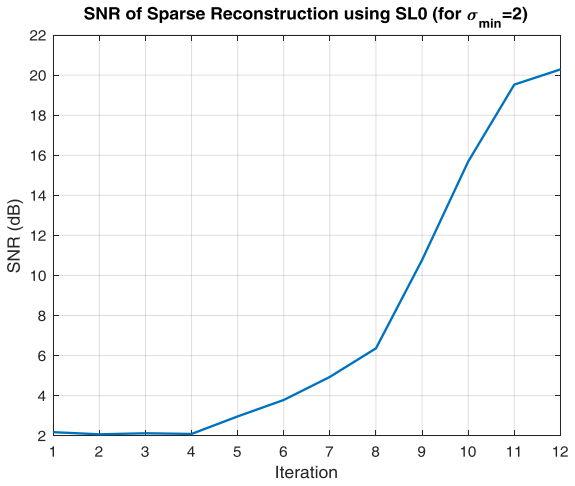


شکل (۱۱): مقایسه پیچیدگی زمانی در شرایطی که تعداد پایه‌های ورودی محدود نباشد.

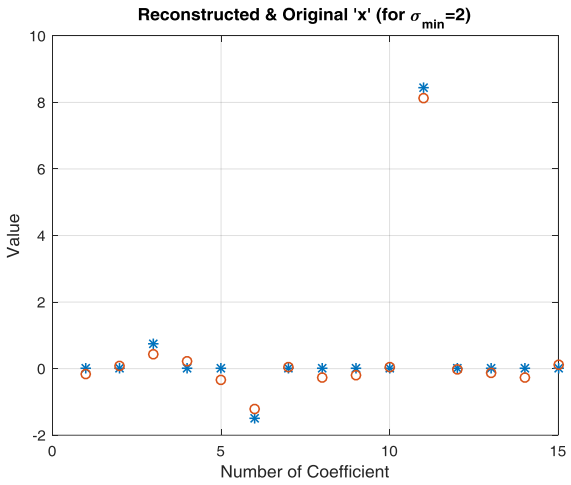
ب) ارزیابی اجرای الگوریتم SL0 با آرایه سیستولیکی

یکی از الگوریتم‌های استفاده‌شده در نمایش (تجزیه) تنک، الگوریتم $SL0$ است. با توجه به رابطه موجود در قسمت (ب.۱.ج) بخش ۲ می‌توان مشاهده کرد که این قسمت به‌دلیل وجود ضرب ماتریس‌های پیچیده اعشاری یکی از مراحل زمان‌بر در فرآیند پردازش الگوریتم $SL0$ است؛ لذا بهره‌گیری از معماری سیستولیک طراحی‌شده، می‌تواند موجب تسریع محاسبات این مرحله از الگوریتم شود. شکل (۱۲) نتیجه اجرای الگوریتم $SL0$ با استفاده از آرایه سیستولیکی در حالتی که دنباله σ برابر [۲،۴،۸] را نشان می‌دهد. همچنین تعداد تکرارهای الگوریتم برای هر مقدار σ برابر $L = 4$ قرار داده شده است؛ درنتیجه تعداد کل تکرارهای الگوریتم برابر ۱۲ تکرار خواهد بود. برای مقایسه میزان خطا از معیار SNR (نسبت سیگنال به نویز) استفاده کرده‌ایم. در اینجا منظور از نویز، درواقع همان خطای بازیابی است که برحسب dB بیان شده است. همان‌طور که مشاهده می‌شود میزان SNR رفته‌رفته زیاد شده است. شکل (۱۳) نیز نتیجه اجرای تکرار آخر را نشان می‌دهد.

با توجه به شکل (۱۳) مشاهده می‌شود که نتیجه تا حد قابل قبولی به پاسخ واقعی نزدیک است. برای رسیدن به‌دقت بیشتر، می‌توان مقدار کمینه σ را کوچک‌تر در نظر گرفته و الگوریتم را مجدداً تکرار کرد.

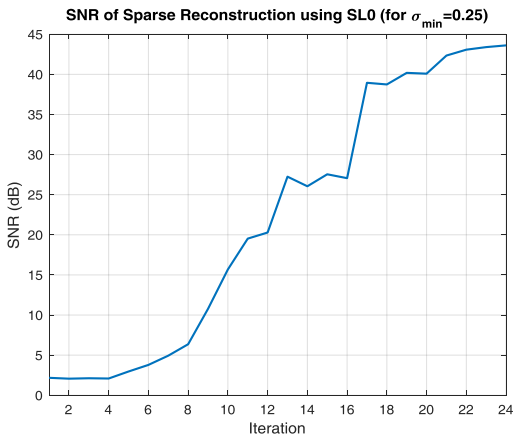


شکل (۱۲): اجرای الگوریتم $SL0$ با استفاده از آرایه سیستولیکی به‌ازای $\sigma = 8, 4, 2$

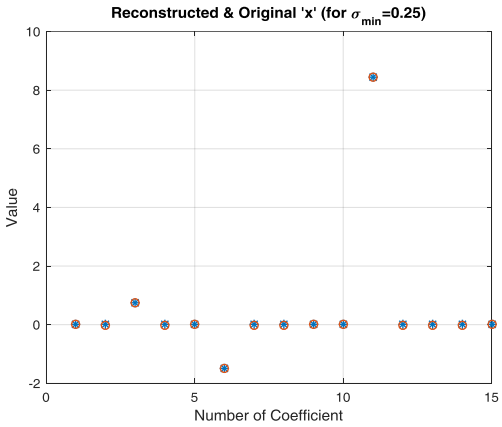


شکل (۱۳): نتایج به‌دست‌آمده از اجرای تکرار آخر با استفاده از آرایه سیستولیکی

شکل (۱۴) نتیجه اجرای الگوریتم $SL0$ با استفاده از آرایه سیستولیکی در حالتی که دنباله σ برابر [۸،۴،۲،۱،۰،۵،۰،۲۵] است را نشان می‌دهد. همچنین تعداد تکرارهای الگوریتم برای هر مقدار σ برابر L=4 قرار داده شده است؛ درنتیجه تعداد کل تکرارهای الگوریتم برابر با ۲۴ خواهد بود. همان‌طور که مشاهده می‌شود میزان SNR در تکرار آخر به حدود ۴۵ dB رسیده است و نسبت به حالت قبل نتیجه بهتری به دست آمده است. نتیجه بازیابی تکرار آخر برای این حالت را در شکل (۱۵) مشاهده می‌شود؛ بنابراین، معماری مبتنی‌بر آرایه سیستولیکی طراحی‌شده می‌تواند موجب تسریع چشم‌گیر الگوریتم $SL0$ در نمایش تنک، با دقت قابل‌قبول شود.



شکل (۱۴): اجرای الگوریتم $SL0$ با استفاده از آرایه سیستولیکی به‌ازای $\sigma = 8, 4, 2, 1, 0.5, 0.25$



شکل (۱۵): نتایج به‌دست‌آمده از اجرای تکرار آخر با استفاده از آرایه سیستولیکی

۵. نتیجه‌گیری

برای ضرب دو ماتریس با ابعاد $N \times N$ به‌صورت تک پردازنده‌ای سری، پیچیدگی زمانی از مرتبه N^3 خواهد بود، به عبارتی ضرب ماتریس با ابعاد $N \times N$ ، N^3 عمل ضرب و $N \times 2(N-1)$ عمل جمع نیاز است. با توجه به این‌که برای انجام هر عمل حداقل ۴ کلاک لازم است پس انجام این عمل با تک پردازنده با $N^3 + 8N^2 + 8N$ کلاک است در صورتی که انجام همین عمل با آرایه سیستولیکی به‌دلیل انجام عملیات به‌صورت موازی و پایپ‌لاین، نیازمند به $(3N - 2)$ کلاک است. زمان پردازش الگوریتم SL0 با آرایه سیستولیکی از O(N3) به O(N) تقلیل می‌یابد. آرایه طراحی شده در این مقاله را می‌توان در کاربردهای یافتن تنک‌ترین جواب در مواردی که بلادرنگ بودن ضروری است از جمله خوشه‌بندی و شناسایی پالس‌های راداری در گیرنده‌های ESM و غیره بکار برد. نتایج حاکی از آن است اجرای الگوریتم مذکور با تک پردازنده با فرض ۴ کلاک برای انجام هر بار معادله بازگشتی کلاکی معادل $N^3 + 9.7N^2 + 3.2N +$ 18لازم دارد؛ در حالی که انجام آن با آرایه سیستولیکی به‌دلیل انجام محاسبات به‌صورت موازی کلاکی معادل$48N + 32$ لازم دارد.

۶. مراجع

[1] Atef Ibrahim, Turki F., Al-Somani, “ Fayez GebaliNew Systolic Array Architecture for Finite Field Inversion,” Canadian Journal of Electrical and Computer Engineering, vol. 40, Issue: 1, pp. 23–30, winter 2017.

[2] D. F. Chiper, A. Cracan, and D. Burdia, “A New Systolic Array Algorithm and Architecture for the VLSI Implementation of IDST Based on a Pseudo-Band Correlation Structure,” Journal of Advances in Electrical and Computer Engineering, vol. 17, pp. 75-80, 2017.

[3] L.-W. Chang and M.-C. Wu, “A unified systolic array for discrete cosine and sine transforms,” IEEE Trans. Signal Process., vol. 39, no. 1, pp. 192–194, 1991.

[4] D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, “A systolic array architecture for the discrete sine transform,” IEEE Trans. Signal Process., vol. 50, no. 9, pp. 2347–2354, Sep. 2002.

[5] L.-W. Chang and S.-W. Lee, “Systolic arrays for the discrete Hartley transform,” IEEE Trans. Signal Process., vol. 39, no. 11, pp. 2411–2418, 1991.

[6] J.-H. Hsiao, L.-G. Ghen, T.-D. Chiueh, and C.-T. Chen, “High throughput CORDIC-based systolic array design for the discrete cosine transform,” IEEE Trans. Circuits Syst. Video Technol., vol. 5, no. 3, pp. 218–225, Jun. 1995.

[7] R. K. Singh, et al., “BioSCAN: A Dynamically Reconfigurable Systolic Array for Biosequence Analysis,” 1996. https://www.researchgate.net/publication/2789050

[8] C.-L. Wang and J.-L. Lin, “Systolic array implementation of multipliers for finite fields GF(2/sup m/),” IEEE Trans. Circuits Syst., vol. 38, no. 7, pp. 796–800, Jul. 1991.

[9] J. G. McWhirter and T. J. Shepherd, “Systolic array processor for MVDR beamforming,” IEE Proceedings F - Radar and Signal Processing, vol. 136, Issue: 2, pp. 75 – 80, April 1989.

[10] R. Baraniuk, “Compressive sensing,” IEEE Signal Processing Magazine, vol. 24, Issue: 4, pp. 118–121, July 2007.

[11] S. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques”, Informatica 31, pp. 249-268, 2007.

[12] R. Gribonval and S. Lesage, “A survey of sparse component analysis for blind source separation: principles, perspectives, and new challenges,” 06 proceedings-14th Eur. Symp., pp. 20-29, 2006.

[13] Y. Li, A. Cichocki, and S. Amari, “Sparse component analysis for blind source separation with less sensors than sources,” ICA, pp. 43-48, 2003.

[14] W. Bajwa, J. Haupt, and A. Sayeed, “Compressed channel sensing: A new approach to estimating sparse multipath channels,” Proceedings of the IEEE, pp. 1058–1076, vol. 98, Issue: 6, 2010.

[15] M. Herman and T. Strohmer, “High-resolution radar via compressed sensing,” IEEE Transactions on Signal Processing vol. 57, Issue: 6, pp. 2275–2284, June 2009.

[16] J. Wright, Y. Ma, J. Mairal, and G. Sapiro, “Sparse representation for computer vision and pattern recognition,” Proceedings of the IEEE, vol. 98, Issue: 6, June 2010, pp. 1031–1044, 2010.

[17] D. Donoho and M. Elad, “Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ1 minimization,” 2003. https://doi.org/10.1073/pnas.0437847100

[18] S. Mallat and Z. Zhang, “Matching pursuits with time-frequency dictionaries,” IEEE Transactions on Signal Processing, vol. 41, Issue: 12, pp. 3397-3415, Dec. 1993.

Journal of Electronical & Cyber Defence
Vol. 7, No. 4, 2020, Serial No. 28

Designing Systolic Array for SL0 Algorithm Implementation

A. Naseri*, R. Jozpiri

*Imam Hossein Comprehensive University

(Received: 02/05/2018, Accepted: 05/03/2019)

ABSTRACT

Systolic architecture is one of most important parallel processing architectures.In the systolic array, ALU units are arranged as an array. This array acts synchronously and executes the recursive equations in parallel by applying the proper input. In this paper, the systolic array for the SL0 is designed and simulated. Simulation results showed that the implementation of this algorithm with a single processor, assuming 4 clocks for executing each recursive equation, requires 4N ^ 3 + 9.7N ^ 2 + 3.2N + 18 clocks, while doing it with a systolic array requires 48n + 32 clocks due to parallel computing and pipelines.

Keywords: Systolic array, Parallel processing, SL0 algorithm, Matrix multiplication

* Corresponding Author Email: anaseri@ihu.ac.ir