# الگوریتم‌ها و محاسبات موازی

مترجم:

مهندس سیامک وطنی

فن‌آوری نوین

| | |
|---|---|
| سرشناسه | جبالی، فایز Gebali, Fayez |
| عنوان و نام پدیدآور | الگوریتم‌ها و محاسبات موازی/[ نویسنده فایز جبالی] ؛ مترجم سیامک وطنی. |
| مشخصات نشر | بابل: فناوری نوین ، ۱۳۹۶. |
| مشخصات ظاهری | ۳۲۶ ص.: مصور ،جدول . |
| شابک | ۳۹۰۰۰۰ ریال: 978-600-7272-06-0 |
| وضعیت فهرست نویسی | فیپا |
| یاداشت | عنوان اصلی :.Algorithms and parallel computing .C,2011 |
| یاداشت | چاپ قبلی: دانشگاه آزاد اسلامی، سازمان چاپ و انتشارات: دانشگاه آزاد اسلامی، واحد فراهان،۱۳۹۳. |
| موضوع | پردازش موازی |
| موضوع | Parallel processing (Electronic computers) |
| موضوع | الگوریتم‌های کامپیوتری |
| موضوع | Computer algorithms |
| شناسه افزوده | وطنی، سیامک، ۱۳۴۹ -، مترجم |
| رده بندی کنگره | الف۲ج/۷QA۷۶/۵۸ ۱۳۹۶ |
| رده بندی دیویی | ۰۰۴/۳۵ |
| شماره کتابشناسی ملی | ۵۱۲۱۹۳۰ |

www.fanavarienovin.net

**فن‌آوری نوین** **بابل، کدپستی ۴۷۱۶۷-۷۳۴۴۸** **تلفن: ۳۲۲۵۶۶۸۷-۰۱۱**

## الگوریتم‌ها و محاسبات موازی

تألیف: سیامک وطنی

نوبت چاپ: چاپ اول

سال چاپ: بهار ۹۷

شمارگان: ۱۰۰۰

قیمت: ۳۹۰۰۰ تومان

شابک: ۰-۰۶-۷۲۷۲-۶۰۰-۹۷۸

نشانی ناشر: بابل، چهارراه نواب، کاظم‌بیگی، جنب مسجد منصور کاظم‌بیگی، طبقه همکف

طراح جلد: کانون آگهی و تبلیغات آبان (احمد فرجی)

تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱۴۲ تلفکس: ۶۶۴۰۰۱۴۴-۶۶۴۰۰۲۲۰

# فهرست مطالب

# مقدمه

## در مورد این کتاب

یک شکاف نرم‌افزاری میان پتانسیل سخت‌افزار و کارایی قابل وصول با به‌کارگیری ابزارهای توسعه نرم‌افزار باقابلیت توازی وجود دارد. این ابزار برای موازی سازی کد برنامه نیاز به مداخله مستقیم برنامه‌نویس دارند. این کتاب قصد دارد تکنیک هایی به برنامه‌نویس ارائه کند تا بتواند توازی را در الگوریتم‌های ترتیبی و یا تکراری کشف نماید. پردازش موازی دیگر مختص سیستم‌های گران‌قیمت خاص که برای گروهای اقلیتی قابل‌دسترس بوده‌اند محسوب نمی‌شود و تمامی سیستم‌های محاسباتی امروزه را در برمی‌گیرد. امروزه می‌توان کامپیوترهای موازی را در لپ تاپ ها، کامپیوترهای رومیزی و سیستم‌های توکار[1] در گوشی‌های هوشمند پیدا کرد. به‌طور سنتی الگوریتم‌ها و برنامه‌هایی که بر روی کامپیوترهای موازی اجرا می‌شدند محدود به پیش‌بینی وضع هوا، شبیه‌سازی تونل باد، محاسبات زیست‌شناسی و پردازش سیگنال[2] بودند. امروزه تقریباً هر برنامه‌ای که بر روی یک کامپیوتر اجرا می‌شود، با پردازشگرهای موازی که قابل‌دسترس در تمامی سیستم‌ها هستند مواجه خواهد شد.

امروزه می‌توان الگوریتم‌های موازی را به‌گونه‌ای طراحی کرد تا بر روی یک کامپیوتر موازی خاص اجراشده و یا آنکه بر روی یک کامپیوتر موازی همه‌منظوره که از فن‌های موازی توسعه نرم‌افزار نظیر: کامپایلرهای موازی، سیستم‌عامل‌های چند نخی[3] و پردازشگرهای سوپر اسکالر[4] استفاده می‌کند، اجرا شود. این کتاب گزینه اول یعنی طراحی معماری‌های پردازشگری موازی تک منظوره برای پیاده‌سازی کلاس خاصی از الگوریتم‌ها را پوشش می‌دهد. ما این سیستم‌ها را هسته‌های شتاب‌دهنده[5] می‌نامیم. این کتاب مبانی درس طراحی و آنالیز الگوریتم‌های موازی را شکل می‌دهد. این درس می‌بایست فصول ۱ تا ۴ را شامل شده و مابقی فصول کتاب به‌عنوان مطالعات موردی انتخاب گردند.

اگرچه فن‌آوری VLSI این اجازه را به ما می‌دهد تا تعداد بیشتری پردازشگر را در یک چیپ[6] مجتمع کنیم، اما برنامه‌نویسی موازی همپای این فن‌آوری پیشرفت نکرده است. یکی از بدیهی‌ترین کاربرد سخت‌افزار موازی، طراحی پردازشگر موازی تک منظوره است که عمدتاً به‌عنوان هسته‌های شتاب‌دهنده در سیستم‌های چندهسته‌ای بکار می‌روند. فراگیر شدن سیستم‌های چندهسته‌ای که در غالب سیستم‌های محاسباتی امروزی وجود دارند و نیاز به الگوریتم‌های موازی ساده در سیستم‌های نظیر: رمزنگاری داده‌ها[7]، پردازش گرافیکی، پردازش سیگنال‌های دیجیتال، فیشلترینگ[8] و سایر کاربردهای عملی، دو دلیلی هستند که موجب توجه بیشتر به این مقوله شده‌اند. راه آسان‌تر در مورد این کتاب پرداختن به مواردی است که جز مباحث آن بشمار نمی‌روند. در حالت کلی این کتاب قصد ندارد مباحث معماری کامپیوتر، کامپیوترهای موازی و یا الگوریتم‌ها را به‌طور کامل موردبررسی قرار دهد. برای بررسی مناسب این سه

---

1. Embdded systems
2. Signal processing
3. Multithreaded
4. Superscalar
5. accelerator cores
6. Chip
7. data encryption
8. filtering

موضوع به سه کتاب درسی خوب نیاز است. منابع استاندارد و البته بی‌نظیر ذیل گزینه‌های مناسبی برای این سه موضوع بشمار می‌روند:

طراحی و سازمان کامپیوتر D.A. Patterson and J.L. Hennessy، معماری کامپیوتر موازی اثر D.E. Culler, J.P. Singh, A. Guptaو ، مقدمه‌ای بر الگوریتم‌ها اثر . H. Cormen, C.E. Leiserson R.L. Rivest. امیدوارم غالب خوانندگان آن‌قدر خوش شانس هستند و مباحث بالا را مطالعه کرده با شند. معمولاً دروس مربوطه بر اساس این سه مرجع تنظیم می‌شوند. اگر کتاب‌های مرجع دیگری در این خصوص را از قلم انداخته باشم، پوزش مرا بپذیرید.

بر این کتاب قصد دارد نحوه طراحی اصولی ساختارهای پردازش موازی تک منظوره، برای پیاده‌سازی الگوریتم‌ها را نشان دهد. تکنیک‌های ارائه‌شده در این کتاب عمومی بوده و قابل‌اعمال به الگوریتم‌های موازی و سایر الگوریتم‌ها می‌باشند.

این کتاب مناسب محققین و فارغ‌التحصیلان رشته‌های مهندسی کامپیوتر، مهندسی برق و علوم کامپیوتر هست. پیش‌نیاز این کتاب یک دانش‌پایه از جبر خطی و پردازش سیگنال‌های دیجیتال است. اهداف این کتاب عبارت‌اند از:

۱. شرح تکنیک‌های مختلف برای بیان یک الگوریتم موازی به صورت گراف وابستگی[1] و یا مجموعه‌ای از ماتریس وابستگی.

۲. کشف الگوی زمان‌بندی برای وظایف پردازشی به‌گونه‌ای که با زمان‌بندی ورودی و خرجی مطابقت دا شته با شند و امکان پایپ‌لاین کردن[2] بعضی از دادها و یا انتشار سایر داده‌ها به صورت Broadcast به تمامی پردازنده‌ها میسر باشد.

۳. کشف الگوی تخصیص وظایف پردازشی به اجزا پردازشگر.

## سازمان فصل و مرور کلی:

فصل ۱) این فصل دو گروه اصلی الگوریتم‌ها: الگوریتم سریال[3]، موازی و تکراری منظم[4] را شرح می‌دهد. طراحی کامپیوتر موازی با توجه به رابطه تنگاتنگ آن با الگوریتم موازی موردبررسی قرار خواهد گرفت. مزایای استفاده از الگوریتم موازی به‌صورت کمی و برحسب معیارهایی نظیر: نسبت تسریع[5] و سربار[6] ارتباط میان پردازشگرها بیان می‌شود. این فصل با توضیح دو کاربرد کامپیوترهای موازی جمع‌بندی می‌گردد.

فصل ۲) در این فصل تکنیک‌هایی نظیر افزایش فرکانس ساعت، موازی کردن واحدهای حساب و منطق[7]، پایپ‌لاین، دستورات خیلی طولانی[8]، محاسبات سوپر اسکالر و چند نخی که به‌منظور افزایش کارایی یک کامپیوتر استفاده می‌شوند، موردبررسی قرار می‌گیرند.

---

[1]. dependence graph [2]. Pipeleine [3]. Serial [4]. Regular iterative [5]. Speedup
[6]. Overhead [7]. arithmetic and logic unit [8]. very long instruction word

فصل ۳) در این فصل ضمن مرور انواع اصلی کامپیوترهای موازی، مباحث حافظه اشتراکی[۱]، حافظه توزیع‌شده[۲]، معماری جریان تک دستور و جریان چند داده[۳]، پردازشگرهای systolic و سیستم‌های چندهسته‌ای بررسی می‌شوند.

فصل ۴) این فصل سیستم‌های چندپردازنده با حافظه اشتراکی را مرور کرده و مباحث انسجام حافظه نهان[۴] و هم‌زمانی فرایندها را شرح می‌دهد.

فصل ۵) این فصل انواع شبکه‌های میان ارتباطی مورداستفاده در پردازنده‌های موازی را مرور می‌کند. انواع شبکه‌های ساده نظیر: باس[۵]، ستاره، حلقه و مش[۶] را شرح می‌دهیم. شبکه‌های کارآمدتری همانند کراس بار[۷] و چندطبقه[۸] نیز موردبررسی قرار خواهند گرفت.

فصل ۶) در این فصل ابزارهای برنامه‌نویسی برای نوشتن کدهای موازی را مرور می‌کند. ابزارهایی نظیر Cilk++، OpenMp و CUDA مرور خواهند شد. تأکید می‌گردد این ابزارها با فرض وابستگی ساده داده‌ها کار می‌کنند. اطمینان از یکپارچگی داده‌ها و زمان‌بندی درست اجرای وظایف بر عهده برنامه‌نویس است. این کتاب به برنامه‌نویس کمک می‌کند تا برای الگوریتم‌های سریال و تکراری به این هدف دست یابد.

فصل ۷) این فصل تکنیک‌های موردی[۹] که برای پیاده‌سازی الگوریتم‌ها بر روی کامپیوترهای موازی مورداستفاده قرار می‌گیرند را، مرور خواهد کرد. تکنیک‌هایی نظیر: زمان‌بندی حلقه‌های مستقل، گسترش حلقه‌های وابسته[۱۰]، باز کردن حلقه‌های وابسته[۱۱]، تقسیم کردن مسئله[۱۲] و راهبرد تقسیم و حل[۱۳] نمونه‌هایی ازاین‌دست محسوب می شوند. استفاده از تکنیک پایپ‌لاین در سطح وظیفه الگوریتم‌ها با کمک روش CORDIC توصیف می‌شوند.

فصل ۸) این فصل الگوریتم NSPA که در مقوله الگوریتم سریال، موازی و یا سریال-موازی نمی‌گنجد را شرح می‌دهد. NSPAها قسمت عمده الگوریتم‌هایی که به‌وضوح موازی نیستند را تشکیل می‌دهند، علاوه بر این NSPAها الگوی وابستگی وظایف را نشان می‌دهند. این فصل یک روش رسمی، بسیار قدرتمند و ساده برای استخراج توازی از یک الگوریتم را شرح می‌دهد. قابلیت مشخص کردن بهترین زمان‌بندی برای یک الگوریتم بر روی یک کامپیوتر موازی، مهم‌ترین مزیت روش رسمی محسوب می‌شود. این روش علاوه بر مزیت قبلی، تعداد پردازنده‌های موازی برای دستیابی به بیشترین نسبت تسریع را مشخص می‌کند. این تکنیک توانایی استخراج پارامترهای مهم کارایی NSPAها نظیر مقدار کار(W)، میزان توازی (P) و عمق (D) را به ما می‌دهد.

فصل ۹) این فصل تبدیل Z را معرفی می‌کند. این تکنیک برای مطالعه نحوه پیاده‌سازی فیلترها، سیستم‌هایی چند نرخی در ماشین‌هایی با پردازش موازی مورداستفاده قرار می‌گیرد. کاربردهایی

---

[1]. shared memory [2]. distributed memory [3]. single instruction multiple data stream
[4]. cache coherence [5]. Bus [6]. Mesh [7]. Multistage [8]. Multistage [9]. Ad hoc
[10]. Dependent loop spreading [11]. dependent loop unrolling [12]. problem partitioning
[13]. divide and conquer

ازاین‌دست معمولاً در حوزه Z مورداستفاده قرار می‌گیرد، و به همین دلیل در این فصل تنها به مطالعه سخت‌افزار و نرم‌افزاری که برای پیاده‌سازی این حوزه استفاده‌شده بسنده می‌کنیم.

فصل ۱۰) این فصل نحوه ساختن گراف وابستگی یک الگوریتم تکراری را شرح می‌دهد. این تکنیک برای الگوریتم‌ها و حلقه‌های تکرار با حداکثر ۳ ایندکس بکار می‌رود. گراف وابستگی امکان زمان‌بندی وظایف و تخصیص خودکار آن‌ها به نخ‌های نرم‌افزاری و پردازنده‌های سخت‌افزاری را فراهم می‌کند.

فصل ۱۱) یک تکنیک تحلیل الگوریتم تکراری که بر اساس مفاهیم محاسبات هندسی و جبر خطی بنانهاده شده‌اند را شرح می‌دهد. در مفهوم عام، این تکنیک می‌تواند الگوریتم‌های تکراری با بیش از سه اندیکس را اجرا کند. فیلترهای دیجیتالی ۲ و ۳ بعدی مثال‌هایی ازاین‌دست محسوب می‌شوند. برای الگوریتم‌هایی از این رده، ابتدا آن را با یک پوش محدب[1] نشان می‌دهیم و سپس یک ماتریس وابستگی به همراه تمامی متغیرهای الگوریتم به آن تخصیص می‌دهیم. فضای خالی در ماتریس امکان استخراج نخ‌های نرم‌افزاری و مؤلفه‌های پردازشی سخت‌افزاری متفاوت به همراه زمان‌بندی مناسب را فراهم می‌آورد.

فصل ۱۲) این فصل ساختارهای پردازشی متفاوت برای پیدا کردن پاسخ تابع ضربه واحد متناهی[2] فیلترهای دیجیتالی یک‌بعدی را نشان می‌دهد. برای شروع، ساختارهای سخت‌افزاری ممکن را با کمک تکنیک‌های هندسی فصل ۱۱ مشخص می‌کنیم. در گام بعدی ساختارهای سخت‌افزاری موازی ممکن با کمک تبدیل Z را در فصل ۹ بررسی خواهیم کرد.

فصل ۱۳ ) این فصل ساختارهای پرداز شی موازی برای یافتن پاسخ ضربه واحد فیلترهای دوبعدی و سه‌بعدی دیجیتالی را نشان می‌دهد. برای این تکنیک از تبدیل Z استفاده خواهیم کرد.

فصل ۱۴ ) این فصل ساختار پردازشی موازی را برای سیستم‌های چند نرخی ضدتداخل[3] و درون یاب[4] نشان می‌دهد. این الگوریتم‌ها کاربردهای زیادی در مخابرات دارند. با کمک تکنیک‌های گراف وابستگی فصل ۱۰ ساختارهای موازی متفاوتی به دست می‌آوریم.

فصل ۱۵ ) این فصل ساختارهای پردازش موازی متفاوتی برای حل مسئله تطبیق الگو[5] ارائه می‌دهد. برای مطالعه این مسئله از تکنیک گراف وابستگی فصل ۱۰ استفاده خواهیم کرد.

فصل ۱۶) این فصل ساختار پردازش موازی برای الگوریتم تخمین حرکت[6] را شرح می‌دهد. الگوریتم اخیر برای فشرده‌سازی ویدئو به کار می‌رود. این موضوع را با استفاده از تکنیک‌های سلسله مراتبی به‌منظور ساده‌سازی مسئله و با کمک گراف وابستگی فصل ۱۰ حل خواهیم کرد.

فصل ۱۷) این فصل ساختار پرداز شی موازی برای ضرب چندجمله‌ای با GF(2) در میدان متناهی[7] را شرح می‌دهد. الگوریتم ضرب با کمک تکنیک گراف وابستگی در فصل ۱۰بررسی می‌شود.

---

[1]. convex hull [2]. infinite impulse response [3]. multirate decimators [4]. Interpolator
[5]. Pattern matching problem [6]. Motion estimation [7]. finite - field

فصل ۱۸) این فصل ساختار پردازشی موازی برای تقسیم چندجمله‌ای بر GF(2) در میدان متناهی را شرح می‌دهد. الگوریتم تقسیم با کمک تکنیک گراف وابستگی در فصل ۱۰بررسی می‌شود.

فصل ۱۹ ) این فصل ساختار پردازشی موازی برای به دست آوردن تبدیل فوریه سریع[1] را شرح می‌دهد. تکنیک‌های پایپ‌لاین برای پیاده‌سازی الگوریتم مرور می گردد.

فصل ۲۰) این فصل سیستم‌های حل معادلات خطی را شرح می‌دهد. این سیستم‌ها را می توان با روش‌های مستقیم و یا غیرمستقیم حل کرد. این فصل نحوه موازی‌سازی تکنیک مستقیم با روش جایگذاری[2] را شرح می‌دهد. یک الگوریتم تبدیل ماتریس چگال[3] به ماتریسی مثلثی با روش دوران گیونز[4] نیز بررسی می‌شود. این فصل روش موازی‌سازی غیرمستقیم SOR را نیز شرح می‌دهد.

فصل ۲۱) این فصل با کمک روش تفاضلی محدود (FDM)[5] به حل معادلات دیفرانسیل با مشتقات جزئی می‌پردازد. این معادلات در مهندسی و علوم بسیار مهم بوده و به منابع محاسباتی زیادی نیاز دارند.

تشکر و قدردانی

در اینجا مراتب امتنان و قدردانی خود را به دکتر M.W.El Kharahi از دانشگاه عین الشمس مصر به خاطر پیشنهادات ارزنده و دلگرمی‌ها در حین آماده کردن این کتاب، اعلام می کنم. در ادامه از همکاران ذیل که کمک آن‌ها در شکل گرفتن این کتاب نقش چشمگیری داشته، سپاسگزارم.

در اینجا مؤلف نام همکاران خود را فهرست کرده که به خاطر طولانی بودن حذف‌شده است.

پیشنهادات و توضیحات

این کتاب طیف وسیعی از تکنیک ها و مباحث مربوط به پردازش موازی را پوشش می‌دهد. به‌احتمال‌زیاد امکان بروز اشتباه و یا از قلم افتادن بعضی از مطالب وجود دارد. ممکن است سایر محققین و یا مهندسین درگیر کار عملی، ایده‌های متفاوتی در ارتباط با مطالب و ساختار این کتاب داشته باشند. از پیشنهادات و توضیحات جدید استقبال خواهیم کرد. اگر خطایی را یافتید، شنیدن آن باعث امتنان ما خواهد بود. همچنین از نظرات جدید در مورد مسائل و تمرین‌ها با ذکر منبع استقبال خواهیم کرد. لطفاً توضیحات و یا گزارش مشکلات را به صورت الکترونیکی به نشانی fayez@uvic.ca و یا به نشانی ذیل به‌صورت فکس و یا نامه ارسال کنید:

Dr. F ayez G ebali
Electrical and Computer Engineering Department
University of Victoria, Victoria, B.C., Canada V8W 3P6
Tel: 250 - 721 - 6509
Fax: 250 - 721 - 6052

---

[1]. fast Fourier transform [2]. forward substitution [3]. Dense matrix [4]. Givens rotation
[5]. finite difference method

# فصل ۱ مقدمه

## ۱–۱. مقدمه

ایده کامپیوتر تک پردازنده خیلی سریع در حال منسوخ شدن است. امروزه برای انجام محاسبات با چنین کامپیوتری، باید راهبردها را تغییر دهیم:

- امکان دسترسی به کارایی مطلوب با یک کامپیوتر تک پردازنده به دلیل توان مصرفی غیرقابل‌قبول پردازنده آن، میسر نیست. راه‌حل عملی برای دستیابی به کارایی موردنظر، استفاده از چندین پردازنده است. ممکن است تعداد کامپیوترهای ساده به چند هزار نیز برسد.
- به‌عنوان یک نتیجه از ویژگی بالا اگر اجرای برنامه بر روی کامپیوتری کند باشد، ممکن است بر روی کامپیوتری دیگر کندتر اجرا شود، چاره کار استفاده از پردازش موازی است.
- ابزارهای برنامه‌نویسی می‌توانند توازی را در یک الگوریتم مفروض در حال توسعه را کشف کنند. وابستگی میان متغیرهای یک الگوریتم می‌تواند **باقاعده** و یا **بی‌قاعده**[1] باشد. در هر دو حالت می‌توان با هم‌زمانی وظایف، سرعت اجرای الگوریتم را با حفظ درستی آن بهبود بخشید.
- **بهینه کردن کارایی**[2] یک کامپیوتر با روش برنامه‌نویسی موازی به سطوح مختلفی نظیر: الگوریتم‌ها، زبان توسعه، سیستم‌عامل، کامپایلر و سخت‌افزار بستگی دارد.
- برای استفاده از مزایای پردازش موازی باید تعداد پردازشگرها و میزان سربار ناشی از ارتباط میان آن‌ها را کاملاً موردتوجه قرار داد. میزان تسریع در برنامه‌هایی با تنگنای محاسباتی به‌سرعت اجرای الگوریتم بر روی پردازنده‌ها بستگی دارد. برای برنامه‌هایی با تنگنای ارتباطی، میزان تسریع به‌سرعت فراهم آوردن داده‌ها برای پردازنده‌ها و استخراج داده‌ها از آن‌ها بستگی دارد.
- حافظه‌های امروزی نسبت به پردازنده‌ها به‌مراتب کندتر هستند و پهنای باند آن‌ها به خواندن و یا نوشتن یک کلمه در سیکل محدود می‌باشند.

---

1. Irregular 2. performance

- امروزه دانشمندان و مهندسین، نیازهای محاسباتی خود را با ماشین‌های در دسترس تطبیق نمی‌دهند، در عوض به‌طور عملی امکان تغییر و تطابق سخت‌افزار محاسباتی با نیازهای آن‌ها وجود دارد.

ازآنجایی‌که سخت‌افزار و نرم‌افزار از هم متأثرند، این کتاب به مبحث الگوریتم‌ها و ساختار سخت-افزاری خاصی که این الگوریتم‌ها را اجرا می‌کنند، تمرکز کرده است. هر برنامه نوشته‌شده درنهایت اجرا می‌شود و اجرای آن به پشتیبانی سخت‌افزاری که توسط پردازنده و سیستم‌عامل فراهم می‌شود، بستگی دارد. به همین دلیل این فصل را با تعاریف شروع می‌کنیم، سپس روش‌های طراحی مربوطه را شرح داده و در ادامه محدودیت‌های طراحی مربوط به این مباحث را بررسی خواهیم کرد.

## ۱-۲. حرکت به سمت برنامه‌نویسی موازی خودکار

همه با فرایند پیاده‌سازی الگوریتم در نرم‌افزار آشنا هستیم. هنگام نوشتن کد نیازی به دانستن جزئیات **ماشین مقصد**[1] نداریم، چراکه این وظیفه به عهده کامپایلر است. اما هنگام نوشتن کد و یا اشکال‌زدایی[2] خروجی یک برنامه با تفکر سیستم تک پردازنده‌ای **و پردازش ترتیبی**[3] عمل می‌کنیم. اما برخلاف تصور ما فرایندهایی که الگوریتم‌ها را در سخت‌افزار و یا نرم‌افزار یک ماشین موازی پیاده‌سازی می‌کنند ارتباط بیش‌تری باهم دارند. شکل ۱-۱ مراحل و یا لایه‌های[4] پیاده‌سازی یک برنامه در سخت‌افزار و یا نرم‌افزار با کمک کامپیوترهای موازی را نشان می‌دهد. برنامه و یا مسئله‌ای که قرار است بر روی **یک پلتفرم محاسباتی موازی**[5] پیاده‌سازی گردد، در لایه پنجم و یا کاربرد تعریف می‌گردد. مشخصات ورودی و خروجی برنامه موردمطالعه نیز تعریف می‌گردد. برای مشخصات بعضی از ورودی‌ها و خروجی‌ها باید محل ذخیره‌سازی داده‌ها و زمان‌بندی مناسب میان داده‌ها را نیز در نظر گرفت. برای راهنمایی توسعه الگوریتم نتایج این لایه به سمت لایه‌های پایین‌تر هدایت می‌شوند.

**لایه ۴، لایه توسعه الگوریتم**، برای پیاده‌سازی برنامه موردنظر محسوب می‌گردد. محاسباتی که برای پیاده‌سازی برنامه موردنیاز است، وظایف الگوریتم و وابستگی میان آن‌ها را تعریف می‌کند. ازآنجایی‌که به‌طور سنتی از اجرای خطی وظایف استفاده می‌کنیم، ممکن است در این مرحله الگوریتمی که برای برنامه موردنظر توسعه داده‌ایم دارای ویژگی توازی و یا فاقد آن باشد. در این مرحله توجهی به زمان‌بندی وظایف و یا تخصیص آن‌ها به پردازنده‌ها نداریم. ممکن است پرداختن به این موضوع‌ها تا حدودی وسوسه‌برانگیز باشد ولی این رویکرد ممکن است با حذف قابلیت توازی، نتیجه برعکس تولید کند. نتیجه

---

1. target computer 2. debugging 3. sequential processing 4. layers
5. parallel computing platform

این لایه، گراف وابستگی، گراف جهت‌دار و یا ماتریس مجاورت برای خلاصه کردن وابستگی وظایف است.

**لایه ۳ و یا لایه توازی**، لایه‌ای است که توازی پنهان در الگوریتم را استخراج می‌شود. این لایه توصیف الگوریتم را از لایه ۴ گرفته و زمان‌بندی **نخ‌ها**[1] و تخصیص آن‌ها به پردازنده‌ها را جهت پیاده‌سازی نرم-افزاری تولید می‌کند. برای پیاده‌سازی در سطح سخت‌افزار، VLSI این لایه را تولید می‌نماید. این لایه در حقیقت زمان‌بندی وظایف و تخصیص آن‌ها به پردازنده‌ها را انجام می‌دهد. این کتاب بر این لایه تأکید می‌کند، همان‌طوری که در شکل نشان داده‌شده این لایه در مستطیلی خاکستری‌رنگ و با گوشه‌های خمیده محصورشده است.

**لایه ۲ و یا لایه کد کردن**، لایه‌ای است که الگوریتم موازی با کمک یک زبان سطح بالا به کد تبدیل می‌شود. زبان انتخابی به پلتفرم محاسباتی موازی مقصد بستگی دارد. شاخه سمت راست شکل ۱-۱ نگاشت الگوریتم به یک پلتفرم محاسباتی موازی همه‌منظوره را نشان می‌دهد. در حقیقت این حالت تلقی ما از برنامه‌نویسی موازی است. برنامه‌نویسی موازی کامپیوترها با ابزاری موسوم به پلتفرم‌های هم‌زمانی آسان‌تر شده است، درواقع این ابزار به برنامه‌نویس کمک می‌کنند تا مدیریت نخ‌ها و زمان‌بندی وظایف اجرایی بر روی پردازنده‌ها را مدیریت کنند. ++Cilk، openMP و CUDA که در فصل ۶ توضیح داده خواهند شد، مثال‌هایی **از پلتفرم‌های هم‌زمانی**[2] می‌باشند.

شاخه سمت چپ شکل ۱-۱ **نگاشت**[3] الگوریتم به یک کامپیوتر موازی خاص نظیر آرایه‌های systolic را نشان می‌دهد. برنامه‌نویس از زبان‌های توصیف سخت‌افزار نظیر VHDL و یا Verilog استفاده می‌کند.

پیاده‌سازی واقعی الگوریتم و یا برنامه بر روی یک پلتفرم موازی در لایه ۱ انجام می‌گیرد. پیاده‌سازی می‌تواند با کمک چند نخی بر روی یک کامپیوتر موازی و یا بر روی سیستمی با چندپردازنده موازی که به‌منظور خاص و با کمک ASIC و یا FPGA ساخته‌شده است، انجام گیرد. منظور ما از برنامه‌نویسی خودکار کامپیوتر موازی چیست؟ در حال حاضر برنامه‌نویسی خودکار سریال را در اختیار داریم. برنامه‌نویس کدی را به زبانی سطح بالا نظیر C، Java و یا FORTRAN نوشته و بدون دخالت وی برنامه ترجمه می‌گردد. به‌عبارتی‌دیگر، برنامه‌نویس نیازی به دانستن جزئیات سخت‌افزار پلتفرم محاسباتی ندارد. اگر برنامه‌نویس دانشی در مورد سلسله‌مراتب حافظه، جزئیات پردازنده و ... نداشته باشد کد سریع به دست

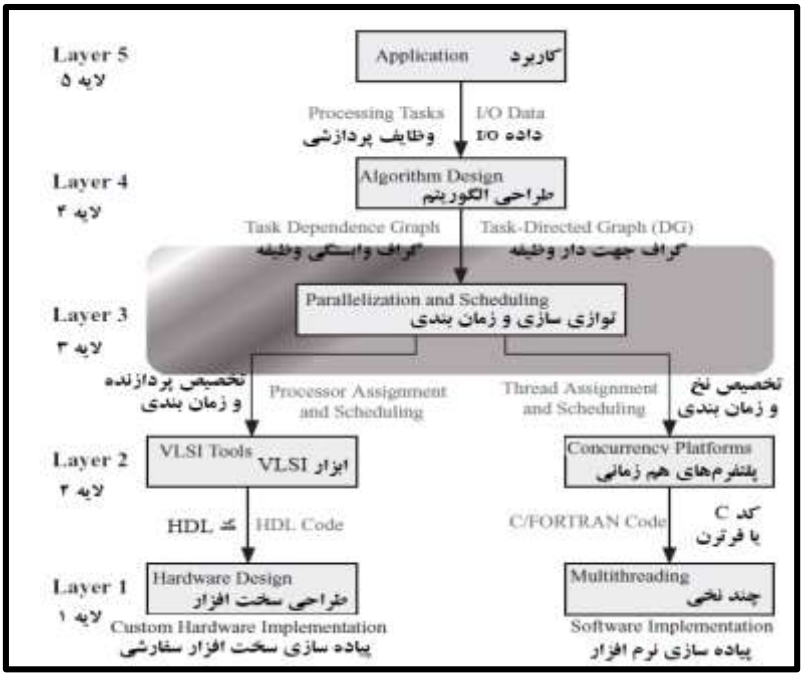
---

[1].threads [2]. Concurrency platform [3]. mapping

خواهد آمد. آیا این موضوع قابل‌تعمیم به تمام کامپیوترهای موازی است؟ کامپایلرهای موازی‌سازی داریم که حلقه‌ای تکرار ساده را در کد پیداکرده و آن را میان پردازنده‌ها توزیع می‌کنند. به هر صورت برنامه‌نویس می‌بایست دانشی درزمینه تعامل میان پردازنده‌ها و زمان مناسب اجرای وظایف برنامه‌ها داشته باشد.

## ۱-۳. الگوریتم‌ها

دیکشنری استاندارد واژگان برق و الکترونیک IEEE الگوریتم را این‌گونه تعریف می‌کند: تعدادی متناهی فرایند و یا قواعد خوش‌تعریف که برای حل یک مسئله بکار می‌روند. در حالت کلی وظایف یک الگوریتم از هم مستقل هستند. بعضی از وظایف را می‌توان به‌صورت هم‌زمان و مابقی را می‌بایست به ترتیب و یکی بعد از دیگری اجرا کرد. با این تعریف یک الگوریتم از دو بخش سریال و موازی تشکیل شده است. به‌جز در موارد خیلی ساده تفکیک الگوریتم‌ها به‌صورت موازی و یا سریال کار دشواری است. در ادامه قادر خواهیم بود این مسئله را به‌صورت کمی بیان کنیم. اگر تعداد وظایف یک الگوریتم W باشد، کار منتسب با این الگوریتم نیز W خواهد بود.




**شکل ۱- ۱ مراحل و یا لایه‌های پیاده‌سازی یک برنامه در سخت‌افزار و یا نرم‌افزار با کمک کامپیوترهای موازی را نشان می‌دهد.**

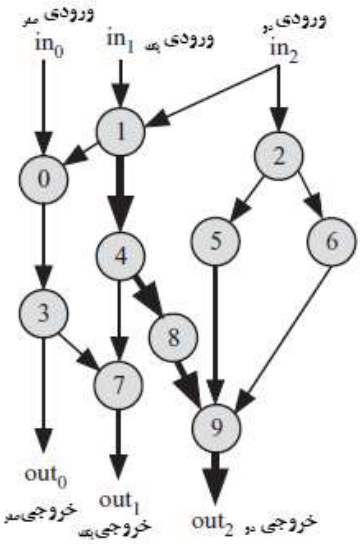
مؤلفه‌های اصلی که یک الگوریتم را تعریف می‌کنند عبارت‌اند از:

- وظایف مختلف.
- وابستگی وظایف برای مواقعی که خروجی یک وظیفه ورودی وظیفه دیگر باشد.
- مجموعه ورودی‌های اصلی که موردنیاز الگوریتم می‌باشند.
- مجموعه خروجی‌های اصلی که توسط الگوریتم تولید می‌شوند.

## ۱-۳-۱. الگوریتم DG

برای نشان دادن وابستگی داده‌ای میان وظایف یک الگوریتم، از نمایش آن به‌صورت DG استفاده می‌شود. از عبارت DG استفاده می‌کنیم تا به این موضوع تأکید کنیم که متغیرهای الگوریتم همانند داده‌ها که با پیکان نمایش داده می‌شوند، میان وظایف در جریان هستند. به بیانی دیگر، **گراف وابستگی**، گرافی است که **یال‌های**[1] آن فاقد پیکان می‌باشند و به همین دلیل، نمایش وابستگی داده‌ها با این گراف کار دشواری محسوب می‌شود.

**تعریف ۱-۱) گراف وابستگی مجموعه‌ای از گره‌ها[2] و یال‌ها است. گره‌ها معرف وظایفی است که توسط الگوریتم اجرا می‌شوند و یال‌ها بیانگر داده‌های هستند که مورداستفاده وظیفه‌ها قرار می‌گیرند. داده‌ها می‌توانند ورودی، خروجی و یا نتایج داخلی[3] باشند.**




**شکل ۱- ۲ مثال از یک گراف جهت‌دار فاقد سیکل (DAG) برای یک الگوریتم.**

---

[1]. Edge [2]. Node [3]. intermediate

دقت کنید یال‌ها در گراف وابستگی فاقد جهت می‌باشند، بنابراین یالی که دو گره را به هم وصل می‌کند، نمی‌تواند معرف ورودی، خروجی و یا وابستگی داده‌ای باشد. یک یال معرف تمامی گره‌های است که یک نمونه از متغیر میان آن‌ها به اشتراک گذاشته‌شده است. این متغیر می‌تواند ورودی، خروجی و یا عملیات I/O که معرف نتایج میانی هستند، باشد.

**تعریف ۱-۲) یک DG مجموعه‌ای از گره‌ها و یال‌های جهت‌دار است. گره‌ها معرف وظایفی[1] است که توسط الگوریتم اجرا می‌شوند و یال‌ها بیانگر وابستگی داده‌های میان وظایف هستند. شروع یال، خروجی یک وظیفه و پایان یال، ورودی وظیفه دیگر است.**

**تعریف ۱-۳) یک گراف جهت‌دار فاقد سیکل[2] (DAG) یک DG است که حلقه ندارد. شکل ۱-۲ نمایش یک الگوریتم با کمک DAG را نشان می‌دهد. یک DG و یا DAG بسته به مبدأ و مقصد یال‌ها دارای سه نوع یال هستند.**

**تعریف ۱-۴) در DG یال ورودی، یالی است که به یک و یا چند گره ختم شده ولی از هیچ گره‌ای آغاز نمی‌شود. این یال معرف یک ورودی از چند ورودی‌ها است. با مراجعه به شکل ۱-۲ متوجه می‌شویم که این الگوریتم دارای سه ورودی بنام‌های $in_0$، $\mathrm{in}_1$ و $\mathrm{in}_2$ است.**

**تعریف ۱-۵) در DG یال خروجی، یالی است یک گره آغازشده ولی به هیچ گره‌ی ختم نمی‌شود. این یال معرف یک خروجی از چند خروجی است. با مراجعه به شکل ۱-۲ متوجه می‌شویم که این الگوریتم دارای سه خروجی بنام‌های $\mathrm{out}_0$، $out_1$ و $out_2$ می‌باشد.**

**تعریف ۱-۶) یال میانی، یالی است یک گره آغازشده ولی به یک و یا چند گره‌ی ختم می‌شود. این یال معرف یک یال میانی از چندین یال‌های میانی است.**

**تعریف ۱-۷) در DG یک گره ورودی، گره‌ی است که تمامی لبه‌های واردشده به آن، لبه‌های ورودی هستند. با مراجعه به شکل ۱-۲ متوجه می‌شویم که گره‌های ۰، ۱ و ۲ گره‌های ورودی هستند. به‌محض فراهم بودن ورودی‌ها، امکان شروع وظایف مرتبط با این گره‌ها میسر است.**

**تعریف ۱-۸) در DG یک گره خروجی، گره‌ی است که تمامی لبه‌های خارج‌شده از آن، لبه‌های خروجی هستند. با مراجعه به شکل ۱-۲ متوجه می‌شویم که گره‌های ۷ و ۹ گره‌های خروجی هستند. دقت کنید در شکل ۱-۲ گره ۳، یک گره خروجی محسوب نمی‌شود، چراکه یکی از یال‌های خروجی آن به‌عنوان یک یال میانی به گره ۷ متصل شده است.**

**تعریف ۱-۹) گره میانی، گره‌ی است حداقل دارای یک یال ورودی و حداقل یک یال خروجی است.**

### ۱-۳-۲) ماتریس مجاورت[3] الگوریتم

یک الگوریتم را می‌توان به‌صورت جبری با ماتریس مجاورت A نشان داد. این ماتریس برای یک الگوریتم با W عدد وظیفه/گره یک ماتریس مربعی W در W با درایه‌های ۰ و ۱ می‌باشد. اگر گره i به خروجی گره j بستگی داشته باشد، a(i, j) = 1 خواهد بود. ازآنجایی‌که هیچ گره‌ی به خروجی خود وابسته نیست، a(i,j) = 0 خواهد شد. ماتریس مجاورت یک **ماتریس نامتقارن**[4] است، چراکه اگر گره i

---

[1]. Task [2]. directed acyclic graph [3]. Adjacency [4]. asymmetric

به گره j وابسته باشد، عکس این موضوع صادق نخواهد بود. برای مثال ماتریس مجاورت شکل ۲ در رابطه ۱-۱ نشان داده‌شده است:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad (1.1)$$

این ماتریس ویژگی‌های جالبی در ارتباط با مباحث ما دارد. گره ورودی i به سطر i مربوط بوده و تمامی عناصر آن ۰ است. گره خروجی j به ستون j مربوط بوده و تمامی عناصر آن ۰ است.

$$\text{Input node i} \Longrightarrow \sum_{j=0}^{W-1} a(i,j) = 0 \qquad ۲-۱$$

$$\text{Output node j} \Longrightarrow \sum_{i=0}^{W-1} a(i,j) = 0 \qquad ۳-۱$$

مابقی گره‌ها همگی داخلی محسوب می‌شوند. ازآنجایی‌که گره‌های ۰، ۱ و ۲ همگی ورودی هستند، عناصر سطرهای ۰، ۱ و ۲ همگی صفر هستند. این مطلب با فونت پهن در این سه سطر مشخص‌شده است. ازآنجایی‌که گره‌های ۷ و ۹ همگی خروجی هستند، عناصر ستون‌های ۷ و ۹ همگی صفر هستند. این مطلب با فونت پهن در این دو ستون مشخص‌شده است. مابقی سطرها و ستون‌ها که هرکدام حداقل یک عنصر غیر صفر دارند، داخلی محسوب می‌شوند. اگر رابطه a(i, j) = 1 برای گره i برقرار باشد به گره j، پدر گره i اطلاق می‌شود.

## ۱-۳-۳. طبقه‌بندی الگوریتم‌ها بر اساس وابستگی وظایف

در حالت الگوریتم‌ها را برحسب وابستگی وظایف می‌توان به شرح زیر دسته‌بندی کرد:

- الگوریتم‌های سریال
- الگوریتم‌های موازی
- الگوریتم‌های سریال - موازی(SPA)[1]
- الگوریتم‌های غیر سریال - موازی(NSPA)[2]

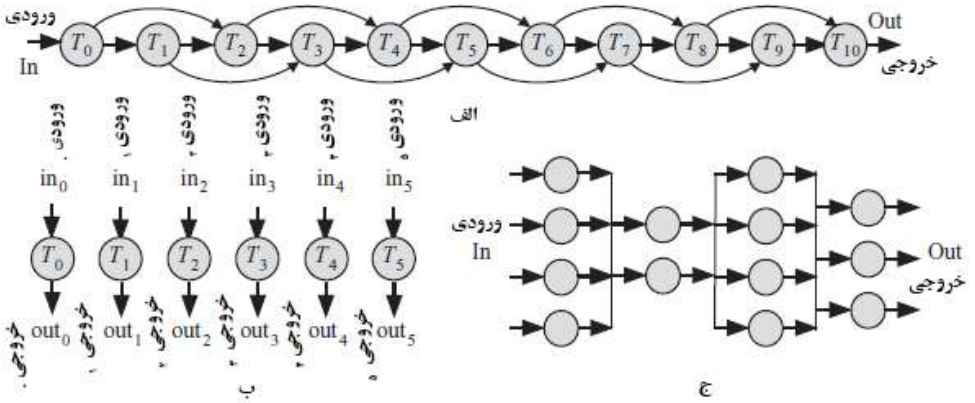
[1]. Serial– parallel algorithms [2]. Nonserial – parallel algorithms

الگوریتم‌های تکراری باقاعده(RIA)[1]

دسته آخر را می‌توان حالت کلی‌تر SPA تصور کرد. لازم به ذکر است که سطح داده و نوع دستورات می‌تواند موجب تغییر کلاس یک الگوریتم به کلاس دیگر شود. برای مثال در جمع دو ماتریس اگر در هرلحظه فقط دو عدد جمع شوند، یک الگوریتم سریال تلقی می‌شود. اگر جمع سطرهای مربوطه در کامپیوترهای متفاوت انجام شود، یک الگوریتم موازی مبتنی بر سطر قلمداد می‌شود. لازم به ذکر است ممکن است تعدادی الگوریتم‌ها شامل سایر الگوریتم‌های دیگر در درون وظایف خود باشند. ممکن است هر پردازنده در هرلحظه فقط دو عدد از دو سطر متناظر را جمع کند، بنابراین الگوریتم موازی، الگوریتم سریال جمع سطر را نشان می‌دهد. در بخش‌های بعدی این دسته‌بندی‌ها را شرح خواهیم داد.

## ۱-۳-۴. الگوریتم‌های سریال

یک الگوریتم سریال، الگوریتمی است که وظایف آن‌ها یکی پس از دیگری و برحسب وابستگی داده‌ها، اجرا می‌شوند. DG مربوط به این الگوریتم‌ها شبیه یک‌رشته طولانی و یا **صفی**[2] از وظایف وابسته است.




**شکل ۱-۳) مثالی از الگوریتم سریال، موازی و سریال - موازی. (الف) الگوریتم سریال. (ب) الگوریتم موازی. (ج) الگوریتم سریال - موازی.**

شکل ۱-۳ الف مثالی از الگوریتم سریال را نشان می‌دهد. الگوریتم نشان داده‌شده برای محاسبه **اعداد فیبوناچی**[3] است. برای محاسبه عدد فیبوناچی $n_{10}$ باید وظیفه $T_{10}$ محاسبه زیر را با شرایط اولیه $n_0 = 0$ و $n_1 = 1$ انجام می‌دهد:

[1].Regular iterative algorithms [2].Queue [3]. Fibonnaci

$$n_{10} = n_8 + n_9 \qquad ۴-۱$$

بدیهی است زمانی می‌توانیم عدد فیبوناچی را محاسبه کنیم که دو عدد فیبوناچی قبلی را محاسبه کرده باشیم.

## ۱-۳-۵ الگوریتم‌های موازی

الگوریتم موازی، الگوریتمی است که وظایف به دلیل استقلال داده‌ها می‌توانند به‌صورت موازی و هم‌زمان اجرا شوند. DG مربوط به چنین الگوریتمی شبیه به یک ردیف پهن از وظایف مستقل است. شکل ۱-۳ ب مثالی از الگوریتم موازی را نشان می‌دهد. یک مثال از پردازش موازی وب سرور است، در این سرور درخواست‌های ورودی مستقل از سایر درخواست‌ها پردازش می‌گردد. مثال دیگر از پردازش موازی سیستم‌عامل چندوظیفه‌ای است، چنین سیستم‌عاملی می‌تواند چندین برنامه نظیر: مرورگر وب، پردازشگر متن و ... را هم‌زمان اجرا کند.

## ۱-۳-۶. SPAها

در یک SPA، وظایف به‌صورت طبقات دسته‌بندی می‌شوند، وظیفه‌های هر طبقه به‌صورت موازی و طبقات به‌صورت سریال اجرا می‌شوند. زمانی که تعداد طبقات ۱ باشد، یک SPA تبدیل به الگوریتمی موازی می‌شود. زمانی که تعداد وظیفه‌ها در هر طبقه ۱ باشد، این الگوریتم به نوع سریال بدل خواهد شد. شکل ۱-۳ ج نمونه‌ای از یک الگوریتم SPA را نشان می‌دهد. الگوریتم CORDIC[1] مثالی از یک الگوریتم SPA است. در این الگوریتم، یک حلقه n بار تکرار شده و در هر دور از حلقه محاسبات زیر انجام می‌شود:

$$\begin{aligned} x_{i+1} &= x_i + \mathrm{m} y_i \delta_i \\ y_{i+1} &= y_i - x_i \delta_i \\ z_{i+1} &= z_i + \theta_i \end{aligned} \qquad ۵-۱$$

x، y و z مقادیری هستند که در هر دور از حلقه تغییر می‌کنند. $\delta_i$ و $\theta_i$ ثابت‌های تکرار بوده و در جدول جست‌وجو ذخیره می‌شوند. پارامتر m متغیر کنترلی بوده نوع محاسبات را مشخص می‌کند. مقدار $\theta_i$ قبل از شروع هر دور حلقه مشخص می‌شود. این الگوریتم محاسبات دیگری نیز انجام می‌دهد که فعلاً در مورد آن‌ها در این قسمت صحبت نمی‌کنیم. برای جزئیات بیش‌تر به فصل ۷ و یا منابع ذکرشده مراجعه کنید.

---

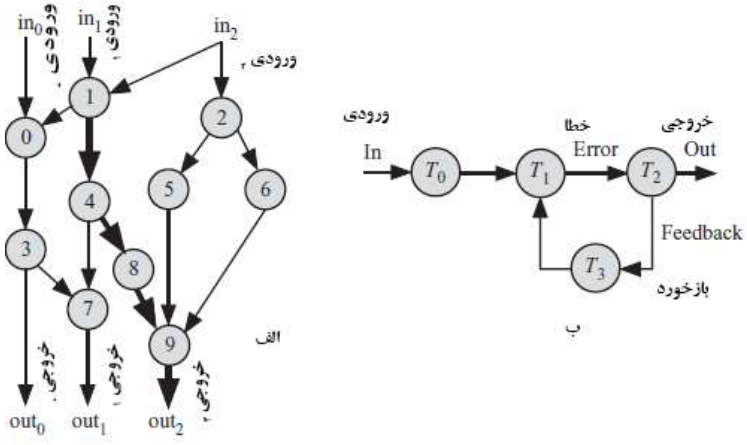
[1]. High - Performance Coordinate Rotation Digital Computer

## ۱-۳-۷) NSPAها

یک الگوریتم NSPA با هیچ‌یک از رده‌بندی‌های بالا مطابقت ندارد. DG این الگوریتم از هیچ الگوی خاصی پیروی نمی‌کند. در ادامه این الگوریتم‌ها را بر اساس وجود حلقه در DG مربوط به الگوریتم، به دو گروه تقسیم خواهیم کرد. بنابراین دو نوع گراف برای الگوریتم NSPA وجود دارد:

- DAG
- گراف جهت‌دار چرخشی (DCG)[1]

شکل ۱-۴ الف مثالی از یک الگوریتم DAG و شکل ۱- ۴ب مثالی از یک الگوریتم DCG است. نوع DCG معمولاً در سیستم‌های کنترل **فیدبک دار**[2] به چشم می‌خورند. ورودی و یا سیگنال وضعیت به وظیفه $T_0$ اعمال می‌شود. وظیفه $T_1$ سیگنال وضعیت و سیگنال خروجی را به‌صورت فیدبک دریافت می‌کند. به خروجی این وظیفه سیگنال خطا اطلاق می‌گردد. این سیگنال به $T_2$ اعمال‌شده و درنهایت سیگنال خروجی تولید می‌شود.




**شکل ۱-۴ مثالی از گراف‌های جهت‌دار برای الگوریتم NSPA. (الف)گراف جهت‌دار فاقد چرخش (DAG). (ب) گراف جهت‌دار چرخشی (DCG).**

گراف NSPA با دو ساختار گره و یال مشخص می‌شود. گره‌ها معرف وظایف الگوریتم و یال‌های جهت‌دار، جریان داده‌ها میان وظایف را مشخص می‌کنند. پیکانی که وارد گره می‌شود، معرف ورودی و پیکانی که از گره خارج می‌شود معرف خروجی است. اگر خروجی تولیدشده $T_i$ مورداستفاده $T_j$ قرار گیرد، گوییم $T_j$ به $T_i$ وابستگی دارد. این موضوع در گراف با پیکانی از گره i به گره j مشخص می‌شود.

---

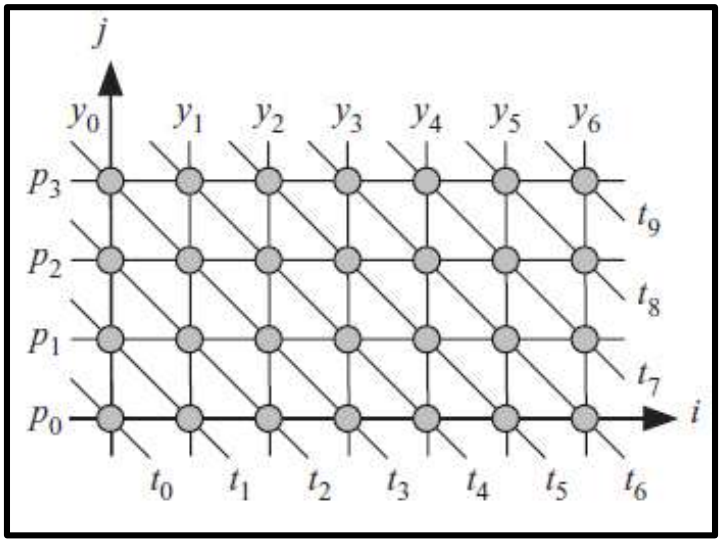
[1]. Directed cyclic graph [2]. Feedback

DG یک الگوریتم سه ویژگی مهم الگوریتم را مشخص می‌کند:

- کار (W) **میزان کارپردازشی**[1] که توسط الگوریتم تکمیل می‌شود را مشخص می‌کند.
- عمق (D) که **عمق بحرانی**[2] هم نامیده می‌شود، به بیشینه ارتفاع و یا عمق میان گره‌های ورودی و خروجی اطلاق می‌گردد.
- توازی (P) که **درجه توازی**[3] هم نامیده می‌شود، حداکثر تعداد گره‌های است که در یک‌زمان به‌صورت موازی پردازش می‌شوند. حداکثر تعداد پردازنده‌های موازی که در یک‌زمان فعال هستند از این مقدار بیش‌تر نخواهد بود، چراکه پردازنده‌های اضافی وظیفه‌ای برای اجرا در اختیار ندارند. در فصل هشتم جزئیات بیش‌تر و نحوه نگاشت یک الگوریتم به کامپیوتر موازی موردبررسی قرار خواهد گرفت.

## ۱-۳-۸) RIAها

Karp مفهوم RIAها را مطرح کرد. ازآنجایی‌که این الگوریتم‌ها کاربرد فراوانی در شاخه‌های مختلف نظیر: پردازش سیگنال، گفتار، تصویر ویدئو، جبر خطی و شبیه‌سازی عددی دارند و به‌صورت ساختارهای شبکه‌ای قابل پیاده‌سازی هستند، ارزش توجهی ویژه دارند. شکل ۱-۵ گراف وابستگی یک RIA را نشان می‌دهد. توجه کنید برای یک RIA گراف DAG را رسم نکرده‌ایم، بلکه به جای آن از مفهوم گراف وابستگی استفاده کرده‌ایم.




**شکل ۱-۵ گراف وابستگی یک RIA برای الگوریتم تطابق الگو.**

---

[1]. processing work [2]. critical path [3]. degree of parallelism

یک گراف وابستگی شبیه یک DAG است، با این تفاوت که یال‌های آن جهت‌دار نیستند. روش به دست آوردن دقیق این گراف را در فصل‌های ۹، ۱۰ و ۱۱ شرح می‌دهیم. در یک RIA وابستگی میان وظایف معرف یک الگوی ثابت است. موازی کردن یک الگوریتم سریال، موازی و حتی SPA یک مسئله ساده محسوب می‌شود، اما استخراج توازی از یک الگوریتم RIA کار آسانی نیست. در حقیقت فصل‌های ۹ تا ۱۱ به نحوه استخراج توازی در این دسته از الگوریتم‌ها اختصاص دارد. یک مثال از الگوریتم RIA ضرب دو ماتریس مطابق الگوریتم ۱-۱ است.

**الگوریتم ضرب دو ماتریس**

```
Require: Input: matrices A and B
1.  for i = 0: I -1 do
2.    for j = 0: J – 1 do
3.        temp = 0
4.        for k = 0 : K -1 do
5.           temp = temp + A(i, k) ×B(k, j)
6.        end for
7.        C(i,j) = temp
8.     end for
9.  enf for
10. RETURN C
```

متغیرهای RIA که در الگوریتم ۱,۱ تعریف‌شده‌اند، معرف وجود وابستگی باقاعده میان اندیس‌های i، j و k در الگوریتم هستند. به‌طور سنتی چنین الگوریتم‌هایی با استفاده از تکنیک گراف وابستگی موردمطالعه قرار می‌گیرند. این روش‌ها ارتباط میان وظایف اجرایی را نشان می‌دهند. زمانی که تعداد اندیس‌های الگوریتم کمتر از سه باشد، گراف وابستگی گزینه مناسبی به نظر می‌رسد. در الگوریتم ضرب ماتریس‌ها با سه اندیکس سروکار داریم، نمایش تصویری چنین گرافی در فضای سه‌بعدی کار دشواری محسوب می‌شود. در این کتاب برای الگوریتم‌هایی با ابعاد بیش‌تر، روش‌های مناسب‌تری مطرح خواهد شد. فصل‌های ۹ تا ۱۱ به مطالعه چنین الگوریتم‌هایی اختصاص دارد.

## ۱-۳-۹) پیاده‌سازی الگوریتم‌ها در کامپیوترهای موازی

در قسمت‌های قبلی انواع گروه‌های الگوریتم بر اساس وابستگی میان وظایف آن‌ها توصیف شد. در این قسمت نحوه پیاده‌سازی الگوریتم‌های مختلف بر روی یک پلتفرم موازی در سطح سخت‌افزار و یا

نرم‌افزار را بررسی خواهیم کرد. به این فرایند، **موازی‌سازی** گویند. راهبرد موازی‌سازی بستگی به الگوریتم موردنظر دارد.

**الگوریتم‌های سریال:** این الگوریتم‌ها که به‌صورت مثال در شکل ۱-۳ الف نشان داده‌شده است، به دلیل ضرورت اجرای ترتیبی وظایف، قابلیت موازی‌سازی ندارند. تنها موازی‌سازی ممکن، زمانی است که یک وظیفه به چندوظیفه کوچک‌تر تقسیم می‌گردد. موازی کردن عملیات ضرب و یا جمع در سطح بیت مثالی ازاین‌دست محسوب می‌شود.

**SPAها:** موازی‌سازی این الگوریتم‌ها که به‌صورت مثال در شکل ۱-۳ ج نشان داده‌شده ، با تخصیص هر وظیفه به یک طبقه ممکن می‌گردد. طبقه در سطح نرم‌افزار یک نخ و در سطح سخت‌افزار با یک مؤلفه پردازشی است. دلیل ماهیت ترتیبی طبقه‌ها، امکان موازی‌سازی‌ها آن‌ها ممکن نیست.

**NSPAها:** روش‌های موازی‌سازی NSPAها در فصل هشتم بررسی خواهد شد.

**RIAها:** روش‌های موازی‌سازی RIAها در فصل ۹ تا ۱۱ بررسی خواهد شد.

## ۱-۴) نکات طراحی محاسبات موازی

این قسمت تعدادی از مهم‌ترین ویژگی‌های طراحی یک سیستم محاسباتی موازی را شرح می‌دهد. برای طراحی یک سیستم محاسباتی موازی باید به گزینه‌های زیادی توجه کرد. طراح می‌بایست معماری پردازنده پایه را به نحوی انتخاب کند، تا بتواند وظایف موردنظر را اجرا کند. این پردازنده می‌تواند یک مؤلفه ساده و یا یک پردازنده سوپر اسکالر که یک سیستم‌عامل چند نخی را اجرا می‌کند، باشد.

پردازنده‌ها باید از طریق یک شبکه درونی با یکدیگر ارتباط برقرار کنند. اگر این شبکه نتواند ارتباط هم‌زمان میان دو پردازنده دلخواه را فراهم کند، می‌تواند به یک **گلوگاه (تنگنای)**[1] سیستم بدل شود. نحوه تبادل داده‌ها باید کاملاً مشخص شود. سیستم باس ساده‌ترین نوع شبکه ارتباطی محسوب می‌شود. داده‌ها به‌صورت کلمه مبادله شده و پردازنده با کمک ساعت سیستم از زمان معتبر بودن داده مطلع می‌شود. امروزه **شبکه‌های میان ارتباطی**[2] جایگزین شبکه‌های قبلی شده‌اند. در این معماری داده‌ها به‌صورت **بسته**[3] بوده و **مسیریاب‌ها**[4] این بسته‌ها را مسیریابی می‌کنند.

داده‌ها و برنامه‌ها می‌بایست در حافظه سیستم ذخیره شوند، طراح می‌تواند ماژول‌های حافظه را به‌صورت اشتراکی میان پردازنده‌ها انتخاب کند و یا به هر پردازنده حافظه انحصاری تخصیص دهد. زمانی

---

1. Bottleneck 2. networks - on - chips (NoC) 3. Packet 4. router

که پردازنده‌ای قصد به اشتراک گذاشتن داده‌ها را داشته باشد، می‌بایستی مکانیسمی برای انجام عملیات خواندن از حافظه و یا نوشتن در ماژول‌های مختلف حافظه فراهم شود. زمانی که یک پردازنده داده‌ای اشتراکی را تغییر می‌دهد، باید به نحوی سایر پردازنده‌ها از این تغییر باخبر شوند تا از مقدار درست داده استفاده کنند.

پیاده‌سازی وظایف و یا برنامه‌ها بر روی یک کامپیوتر موازی نیز با گزینه‌های مختلف طراحی درگیر است. فرایند تقسیم کردن وظایف، برنامه اصلی را به بخش‌های کوچک‌تری تقسیم کرده و هر بخش را به یک پردازنده تخصیص می‌دهد. سطح تقسیم‌بندی میزان حجم کار هر پردازنده را مشخص می‌کند. تقسیم‌بندی Coarse grain قسمت‌های بزرگ‌تری را به هر پردازنده تخصیص می‌دهد. تقسیم‌بندی Fine grain قسمت‌های کوچک‌تری را به هر پردازنده تخصیص می‌دهد. این بخش‌ها می‌توانند به فرم فرایندهای نرم‌افزاری و یا نخ‌های مجزا باشند. برنامه‌نویس و یا کامپایلر در مورد این تقسیم‌بندی تصمیم می‌گیرند. برای درستی اجرای برنامه و جامعیت داده‌ها، برنامه‌نویس و سیستم‌عامل باید از هم‌زمانی درست اجرای وظایف اطمینان حاصل کنند.

## ۱–۵. الگوریتم‌های موازی و معماری‌های موازی

ارتباط تنگاتنگی میان الگوریتم‌های موازی و معماری‌های موازی وجود دارد. تفکر در مورد یک الگوریتم موازی بدون توجه به سخت‌افزار موازی که قصد اجرای آن را دارد و یا تفکر در مورد یک سخت‌افزار موازی بدون توجه به نرم‌افزار موازی مربوطه، روش درستی نیست. در یک سیستم محاسباتی با استفاده از تکنیک‌های سخت‌افزاری و نرم‌افزاری امکان پیاده‌سازی توازی مختلفی وجود دارد:

- **توازی در سطح داده**[1]: زمانی که عملی به‌طور هم‌زمان به روی چندین داده اعمال شود، با این سطح از توازی سروکار داریم. جمع، ضرب و تقسیم اعداد دودویی به‌صورت بیت‌های موازی، آرایه‌ای از پردازنده‌های برداری و آرایه‌های systolic مثال‌هایی ازاین‌دست محسوب می‌شوند.
- **توازی در سطح دستورالعمل** (ILP)[2]: زمانی که بیش از یک دستور به‌طور هم‌زمان در پردازنده‌ای اجرا شود، با توازی در سطح دستور سروکار داریم. تکنیک پایپ لاین مثالی ازاین‌دست محسوب می‌شود.

[1]. Data - level parallelism [2]. Instruction - level parallelism (ILP)

**توازی در سطح نخ** (TLP)[1]: یک نخ قسمتی از برنامه است که منابع پردازنده را با سایر نخ‌ها به اشتراک می‌گذارد. به یک نخ گاهی اوقات واژه فرایند سبک هم اطلاق می‌شود. در TLP نخ‌های مختلف نرم‌افزاری به‌طور هم‌زمان بر روی یک و یا چندپردازنده اجرا می‌شوند.

**توازی در سطح فرایند**[2]: فرایند یک برنامه در حال اجرا بر روی یک کامپیوتر است. یک فرایند منابع کامپیوتری نظیر حافظه و ثبات‌ها را برای خود رزرو می‌کند. این توازی حالت کلاسیک سیستم‌های محاسباتی چندوظیفه‌ای و اشتراک زمانی محسوب می‌شود، در این سیستم‌ها چندین برنامه به‌طور هم‌زمان بر روی یک و یا چند ماشین اجرا می‌شوند.

## ۱–۶) ارتباط الگوریتم موازی و معماری موازی

دیکشنری استاندارد واژگان برق و الکترونیک IEEE، توازی در نرم‌افزار را این‌گونه تعریف می‌کند: انتقال، **رویداد**[3] و یا پردازش هم‌زمان قسمت‌های **منفردی**[4] از یک واحد کل نظیر بیت‌های یک کاراکتر و یا کاراکترهای یک کلمه که با کمک امکانات مجزایی که برای هر بخش انجام می‌گیرد. بنابراین اگر بیش از دو بخش یک الگوریتم به‌طور هم‌زمان و مستقل بر روی سخت‌افزار انجام گیرند، گوییم این الگوریتم موازی است. بنابراین، تعریف الگوریتم موازی به‌طور ضمنی وجود سخت‌افزار مربوطه را تداعی می‌کند. این موضوع به این نکته اشاره دارد که رابطه تنگاتنگی بین یک نرم‌افزار موازی و سخت‌افزاری که آن را اجرا می‌کند، وجود دارد. اجرای قسمت‌های مختلف می‌تواند به‌صورت نخ‌ها و یا فرایندها در نرم-افزار و یا بر روی پردازنده‌های مختلف در سخت‌افزار انجام گیرد. زمانی که در یک کد با حلقه‌های FOR و یا WHILE مواجه می‌شویم، به‌راحتی می‌توانیم قسمت‌های موازی را استخراج کنیم. فراهم آوردن وظایف کافی برای مشغول کردن پردازنده‌ها به عهده برنامه‌نویس، کامپایلر و یا سیستم‌عامل است. مثال‌های متنوعی در شاخه‌های مختلف وجود دارد:

- محاسبات علمی نظیر شبیه‌سازی فیزیکی، حل معادلات دیفرانسیل، شبیه‌سازی تونل باد، شبیه‌سازی آب‌وهوا.
- گرافیک کامپیوتری نظیر پردازش تصویر، فشرده‌سازی ویدئو و Ray Tracing.

---

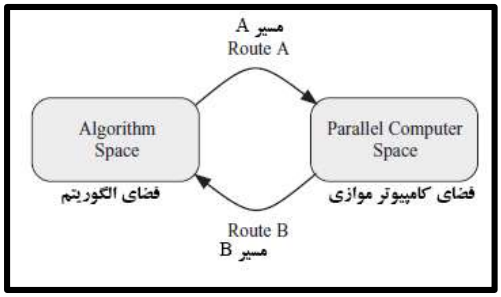
[1]. Thread - level parallelism (TLP) [2]. Process - level parallelism [3]. occurrence, [4]. individual

- تصور برداری پزشکی نظیر MRI [1] و CT [2].

به هر حال تعداد زیادی الگوریتم وجود دارند که قابلیت توازی ندارند، موارد چون داده‌های بر خط پزشکی[3]، بانکداری الکترونیکی[4]، داده‌کاوی[5]، انبار داده‌ها[6] و بازیابی پایگاه داده‌ها[7] مثال‌های ویژه‌ای در دنیای فن‌آوری اطلاعات محسوب می‌شوند. توسعه معماری کامپیوتر و سخت‌افزار، چالشی برای افزایش سرعت برنامه‌های مختلف دنیای فن‌آوری محسوب می‌شود.

## ۱– ۷) پیاده‌سازی الگوریتم‌ها و یا یک مسئله دوطرفه

شکل ۱-۶ یکی از مطالب موردعلاقه ما را نشان می‌دهد. در سمت چپ تصویر فضای الگوریتم‌ها و در سمت راست معماری‌های موازی که این الگوریتم‌ها را اجرا می‌کنند، قرار دارد. مسیر A حالتی است که برای یک الگوریتم داده‌شده به دنبال یک سخت‌افزار موازی و یا آرایه‌ای از پردازنده‌ها هستیم تا ضمن اجرای صحیح الگوریتم، کارایی موردنظر و قیود سیستمی[8] را لحاظ کند. به‌عبارتی‌دیگر، برای یک الگوریتم موازی داده‌شده، سخت‌افزار مناسبی وجود دارد؟




**شکل ۱-۶ دو مسیری که الگوریتم‌های موازی و معماری‌های موازی را به هم ربط می‌دهد.**

مسیر B حالت کلاسیکی است که یک معماری موازی و یا چندهسته‌ای در اختیار داریم و به دنبال بهترین روش برای پیاده‌سازی یک الگوریتم بر روی این سخت‌افزار هستیم تا کارایی موردنظر و قیود سیستم را تأمین کند. به‌عبارت‌دیگر، چگونه می‌توان وظایف مختلف یک الگوریتم موازی را به پردازنده‌های مختلف تخصیص داد؟ این سؤال یکی از مباحث اصلی برنامه‌نویسی موازی با استفاده از تکنیک‌های طراحی چند نخی محسوب می‌شود. این فرایند توسط برنامه‌نویس، کامپایلر و یا سیستم‌عامل انجام می‌گیرد.

---

1. magnetic resonance imaging
2. Computerized tomography
3. online medical data
4. online banking
5. data mining
6. data warehousing
7. database retrieval
8. system constraints

برای حرکت در مسیر A و یا B با موارد زیر سروکار خواهیم داشت:

- نگاشت وظایف به فرایندها.
- زمان‌بندی اجرای وظایف به‌گونه‌ای که با وابستگی داده‌ها و نیازمندی‌های I/O مطابقت داشته باشد.
- مشخص کردن ارتباطات داده‌ای میان پردازنده‌ها و I/O.

## ۱–۸. اندازه‌گیری سودمندی محاسبات موازی

در این قسمت نتایج و منافعی که با استفاده از پردازش موازی حاصل می‌شود را مرور می‌کنیم. اما در ابتدا تعدادی از پارامترهای کلیدی موردمطالعه را شناسایی می‌کنیم.

### ۱–۸–۱. پارامتر تسریع

میزان کارایی محاسبات موازی با مقایسه زمان اجرای یک وظیفه بر روی یک سیستم تک پردازنده و زمان اجرای همان وظیفه بر روی یک سیستم N پردازنده اندازه‌گیری می‌شود. نسبت تسریع استفاده از N پردازنده موازی با رابطه ۱–۶ مشخص می‌شود:

$$S(N) = \frac{T_p(1)}{T_p(N)} \qquad ۱–۶$$

در این رابطه $T_p(1)$ زمان اجرای الگوریتم بر روی یک پردازنده و $T_p(N)$ زمان پردازش بر روی N پردازنده است. در شبیه‌سازی ایدئال یک الگوریتم موازی کامل، و با صرف‌نظر کردن زمان ارتباط میان پردازنده‌ها و حافظه، رابطه $T_p(N) = T_p(1)/N$ برقرار می‌شود. این رابطه نتیجه زیر را به دنبال دارد:

$$S(N) = N \qquad ۱–۷$$

به چند دلیل که متعاقباً در این کتاب بررسی خواهیم کرد، در دنیای واقعی بندرت به این نسبت خطی می‌رسیم.

### ۱–۸–۲. سربار ارتباطات

در سیستم‌های تک پردازنده و سیستم‌های محاسباتی موازی، همواره به خواندن از حافظه و یا نوشتن در حافظه نیاز داریم. به دلیل تفاوت سرعت حافظه با پردازنده، ارتباط میان این دو زمان‌بر است. در سیستم‌های موازی، ارتباط میان پردازنده‌ها و تبادل داده‌ای به‌صورت انتقال داده و پیام از طریق شبکه میان ارتباطی انجام می‌گیرد. ارتباط میان پردازنده‌ها با مشکلاتی مواجه خواهد شد:

**تأخیر شبکه میان ارتباطی**: ارسال داده از طریق شبکه میان ارتباطی از تأخیر انتشار بیت، تأخیر انتقال داده و پیام، تأخیر در صف و ... رنج می‌برد. این فاکتورها به توپولوژی شبکه، اندازه داده ارسالی و سرعت انتشار شبکه بستگی دارند.

**پهنای باند حافظه**: فارغ از ظرفیت حافظه، دسترسی به محتویات حافظه از طریق یک پورت[1] ساده انجام می‌گیرد. این پورت در هر سیکل دسترسی به حافظه، اجازه انتقال یک کلمه به درون حافظه و یا انتقال به خارج از حافظه را می‌دهد.

**تصادم حافظه**[2]: این پدیده زمانی رخ می‌دهد که بیش از یک پردازنده قصد دسترسی به یک ماژول از حافظه را داشته باشند. برای آنکه در هرلحظه فقط یک پردازنده به حافظه دسترسی داشته باشد، باید یک قابلیت حکمیت[3] فراهم گردد.

**دیوار حافظه**[4]: سرعت جابجایی داده‌های حافظه خیلی از سرعت پردازش آن‌ها پایین‌تر است. برای حل این مشکل از سلسله‌مراتب حافظه مطابق نمودار زیر استفاده می‌شود:

register↔cache↔RAM↔electronic disk↔magnetic disk↔optic disk

هنگام پردازش یک الگوریتم بر روی یک سیستم با پردازنده موازی با تأخیرهای متعددی مطابق جدول ۱-۱ مواجه خواهیم شد.

جدول ۱-۱ تأخیرهای درگیر در ارزیابی یک الگوریتم در یک سیستم موازی.

| عملیات | نماد | توضیحات |
|---|---|---|
| خواندن از حافظه | $T_r(N)$ | خواندن از حافظه اشتراکی با N پردازنده |
| نوشتن در حافظه | $T_w(N)$ | نوشتن در حافظه اشتراکی با N پردازنده |
| ارتباط | $T_c(N)$ | تأخیر ارتباطی دو پردازنده در سیستمی متشکل از N پردازنده |
| پردازش داده | $T_p(N)$ | زمان پردازش الگوریتم با N پردازنده |

## ۱-۸-۳. تخمین فاکتور تسریع و سربار ارتباط

فرض کنید یک الگوریتم موازی متشکل ازN وظیفه مستقل داریم، این الگوریتم می‌تواند بر روی یک پردازنده و یا N پردازنده اجرا شود. در حالت ایدئال به علت استقلال وظایف، جابجایی داده‌ها میان پردازنده‌ها و حافظه صورت می‌گیرد. در این شرایط روابط زیر را خواهیم داشت:

$$T_p(1) = N\,\tau_p \qquad ۸-۱$$

[1]. Port [2]. Memory collision [3]. Arbitration [4]. Memory wall

$$T_p(\mathrm{N}) = \tau_p \qquad ۹-۱$$

زمان لازم برای خواندن داده‌های ورودی الگوریتم در یک سیستم تک پردازنده با رابطه زیر محاسبه می‌شود:

$$T_r(1) = \mathrm{N}\ \tau_m \qquad ۱۰-۱$$

$\tau_m$ زمان موردنیاز برای دسترسی به یک بلوک از حافظه است. در این رابطه فرض کرده‌ایم هر وظیفه به یک بلوک از حافظه و N وظیفه به N بلوک از حافظه نیاز دارند. زمان لازم برای خواندن و یا نوشتن از حافظه با رابطه زیر محاسبه می‌شود:

$$T_r(\mathrm{N}) = \alpha\, T_r(\mathrm{N}) = \alpha \mathrm{N}\ \tau_m \qquad ۱۱-۱$$

$\alpha$ فاکتوری است که محدودیت‌های دسترسی به حافظه اشتراکی را توصیف می‌کند. زمانی که هر پردازنده یک کپی از داده‌های خود را در اختیار داشته باشد، $\alpha = \frac{1}{N}$ خواهد شد. زمانی که داده‌ها در یک حافظه مرکزی توزیع شوند $\alpha = 1$ می‌شود. در بدترین حالت، یعنی زمانی که تمامی پردازنده‌ها به داده نیاز داشته باشند، نامساوی $\alpha > N$ برقرار است. موارد فوق را می‌توان مطابق رابطه زیر جمع‌بندی کرد:

$$T_r(\mathrm{N}) \begin{cases} = \tau_m & \text{حافظه توزیع باشد} \\ = N\tau_m & \text{حافظه اشتراکی و بدون تصادم باشد} \\ > N\tau_m & \text{حافظه اشتراکی و با تصادم باشد} \end{cases} \qquad 12-1$$

نوشتن در حافظه ممکن است به دلیل نیاز دسترسی سایر پردازنده‌ها به حافظه با مشکل تصادم مواجه شود:

$$T_w(1) = \mathrm{N}\ \tau_m \qquad ۱۳-۱$$

$$T_w(\mathrm{N}) = \alpha\, T_w(1) = \alpha \mathrm{N}\ \tau_m \qquad ۱۴-۱$$

برای یک پردازنده زمان کامل کردن یک وظیفه با در نظر گرفتن سربار دسترسی به حافظه از رابطه زیر محاسبه می‌شود:

$$T_{total}(1) = T_r(1) + T_p(1) + T_w(1) = \mathrm{N}(2\tau_m + \tau_p) \qquad ۱۵-۱$$

اکنون رابطه تسریع با در نظر گرفتن سربار ارتباطی مطابق رابطه‌های زیر بیان می‌شود:

$$T_{total}(\mathrm{N}) = T_r(\mathrm{N}) + T_p(\mathrm{N}) + T_w(\mathrm{N}) = 2\mathrm{N}\alpha\tau_m + \tau_p \qquad ۱۶-۱$$

$$S(\mathrm{N}) = \frac{T_{total}(1)}{T_{total}(\mathrm{N})} = \frac{2\mathrm{N}\alpha\tau_m + N\tau_p}{2\mathrm{N}\alpha\tau_m + \tau_p} \qquad ۱۷-۱$$

$$R = \frac{\tau_m}{\tau_p} \qquad ۱۸–۱$$

در رابطه بالا مقدار R نسبت عدم **تطابق**[1] حافظه نامیده می‌شود و به‌صورت نسبت زمان دسترسی به یک بلوک از حافظه به زمان پردازش آن تعریف می‌گردد. بسته به تقسیم‌بندی زیر وظایف تحت پردازش و سرعت حافظه انتظار داریم مقدار $\tau_p$ به‌مراتب از $\tau_m$ کوچک‌تر باشد. معادله ۱-۱۷ را می‌توان به فرم زیر نوشت:

$$S(N,R) = \frac{2\alpha NR + N}{2\alpha N\tau_m + 1} \qquad ۱۹–۱$$

رابطه ۱–۱۷ تأثیر مقادیر N و R بر میزان تسریع برای مواقعی که $\alpha = 1$ را نشان می‌دهد. نتایج شبیه-سازی عددی نشان می‌دهد تأثیر متغیر $\alpha$ به‌اندازه تأثیر مقادیر N و R نیست. اگر در رابطه ۱–۱۷ $1 >> RN$ باشد، به تسریع کامل خواهیم رسید. رابطه تسریع در این حالت شبیه حالتی است که سربار ارتباطی ناچیز فرض شده بود. این حالت برای الگوریتم‌های موازی ساده که در فصل ۷ بررسی می‌شوند، روی می‌دهد. دقت کنید برای مواقعی که $RN>0.1$ باشد، تسریع به‌سرعت کم خواهد شد. برای زمانی که $R = 1$ باشد، با یک مسئله از نوع تنگنای ارتباطی[2] مواجه هستیم و در این حالت تمام مزایای توازی از دست می‌رود. این موضوع به اهمیت طراحی حافظه و ارتباط میان پردازنده‌ها و نخ‌ها تأکید می‌کند. در فصل سوم که پردازنده‌های چندهسته‌ای را بررسی خواهیم کرد، خواهیم دید که تمامی پردازنده‌ها در یک چیپ قرار دارند. این ویژگی موجب افزایش سرعت ارتباط میان پردازنده‌ها نسبت به زمانی که در چیپ‌های مختلفی دارند، می‌شود. مقدار R با افزایش درجه چندهسته‌ای کاهش می‌یابد، و به همین دلیل سیستم‌های چندهسته‌ای مقدار R کمی دارند.

---

[1]. memory mismatch ratio [2]. communication - bound problem