دانشگاه صنعتی امیرکبیر
( پلی تکنیک تهران )
دانشکده مهندسی کامپیوتر

# برنامه‌نویسی چندهسته‌ای

## تمرین دوم: آشنایی با الگوریتم‌های تجزیه در برنامه‌نویسی چندهسته‌ای

**رادین شایانفر**

**شماره دانشجویی: ۹۷۳۱۰۳۲**

**بهار ۱۴۰۰**

**۱. ۱ و ۲)** این مسئله را به ۲ روش می‌توانیم تجزیه و حل کنیم:

**۱)** تجزیه داده ورودی به r بخش (Data decomposition): در این حالت ابتدا آرایه ورودی را به r بخش مساوی تقسیم می‌کنیم. سپس هر بخش را به یک تسک می‌دهیم. وظیفه هر تسک (که مجموعا r تسک داریم)، ایجاد r سبد برای داده‌ی خودش و محاسبه‌ی اعضای هر سبد است. پس از آن هر تسک i، داده‌های سبد iام تسک‌های دیگر را می‌گیرد و همه را در یک سبد جمع میکند. در آخر مانند نسخه سریال bucket sort، سبدهایی که دست تسک‌ها هست به یک دیگر متصل می‌شوند.

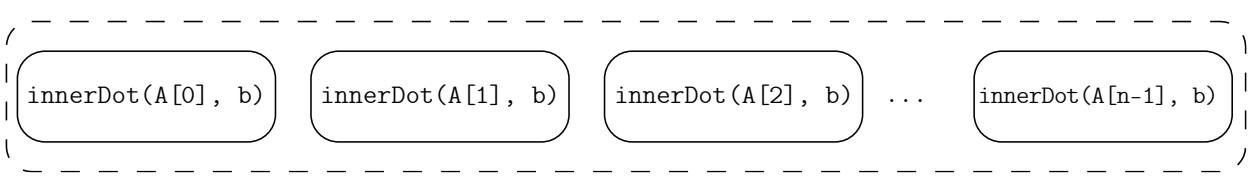
**۲)** تجزیه داده خروجی به r بخش (Task decomposition): در این روش r تسک می‌سازیم. سپس هر تسک روی کل داده‌ی ورودی، داده‌های مربوط به سبد خود را جمع‌آوری می‌کند. در آخر مانند نسخه سریال، سبدهایی که دست تسک‌ها هست به یک دیگر متصل می‌شوند.

**۳)** پاسخ این سوال بستگی به n (طول آرایه) و r (تعداد سبدها) دارد.

در صورتی که **r (تعداد سبدها) بزرگ** باشد، در روش (۱) به دلیل اینکه $r^2$ تا سبد در مجموع توسط نخ‌ها تولید می‌شود، در هنگام ارسال سبدها برای نخ‌های دیگر، کارایی کاهش می‌یابد و استفاده از **روش (۲)** مناسب‌تر است.

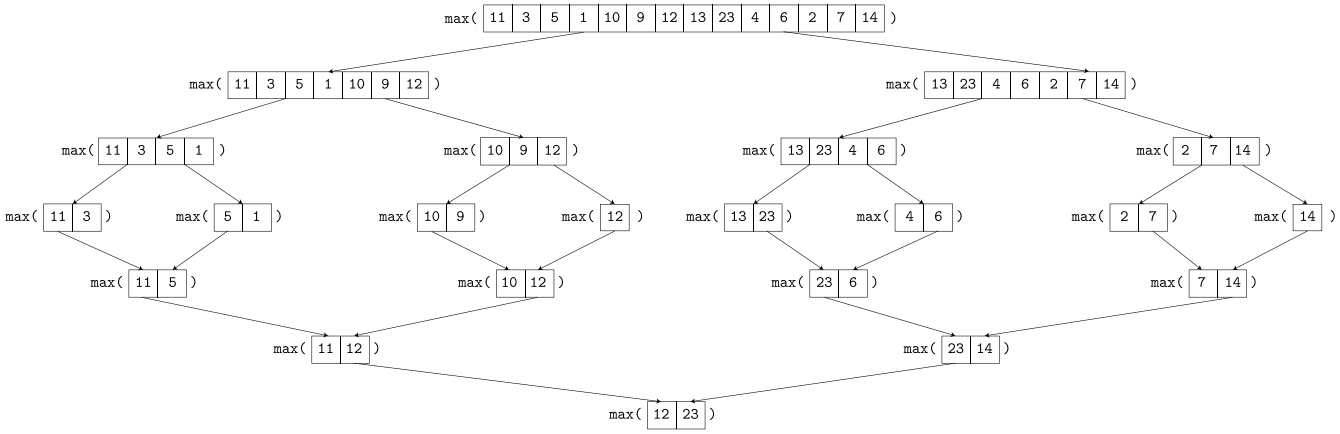
اما اگر **n (طول آرایه) بزرگ** باشد، در روش (۲) هر تسک باید کل آرایه ($O(n)$) را برای ساخت سبدها پیمایش کند و این باعث پایین آمدن کارایی می‌شود. در نتیجه **روش (۱)** با شکاندن n و تقسیم آن بین هسته‌ها می‌تواند در این حالت بهتر عمل کند.

**۲.** در این مسئله ضرب هر سطر در بردار b می‌تواند مستقل از سطرهای دیگر انجام شود. در نتیجه گراف وظایف هیچ یالی ندارد و به شکل زیر خواهد بود.



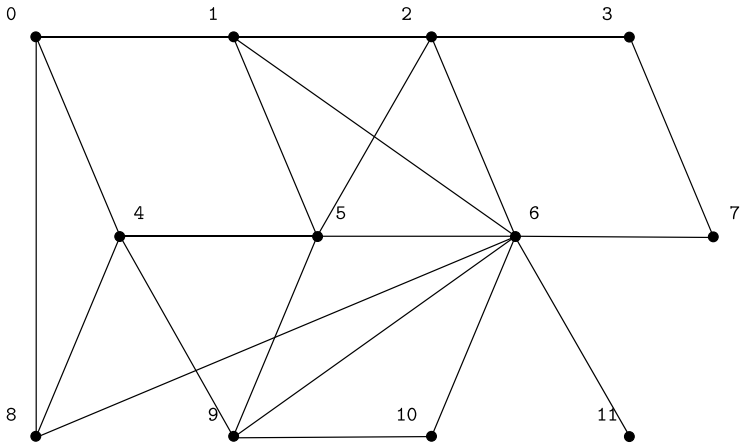

**۳.** ابتدا آرایه را نصف می‌کنیم. سپس روی هر قسمت آن تابع max را به عنوان یک تسک جدید صدا می‌زنیم. در نتیجه آن قدر عمل نصف شدن اتفاق می‌افتد تا اندازه هر بخش دو یا یک شود. سپس هر دو تکه‌ی نصف شده را بینشان max می‌گیریم تا max کل آرایه پیدا شود. گراف وظایف آن را در زیر می‌بینیم.

همان‌طور که می‌بینیم در نهایت عدد ۲۳ به عنوان مقدار max آرایه برگردانده می‌شود.

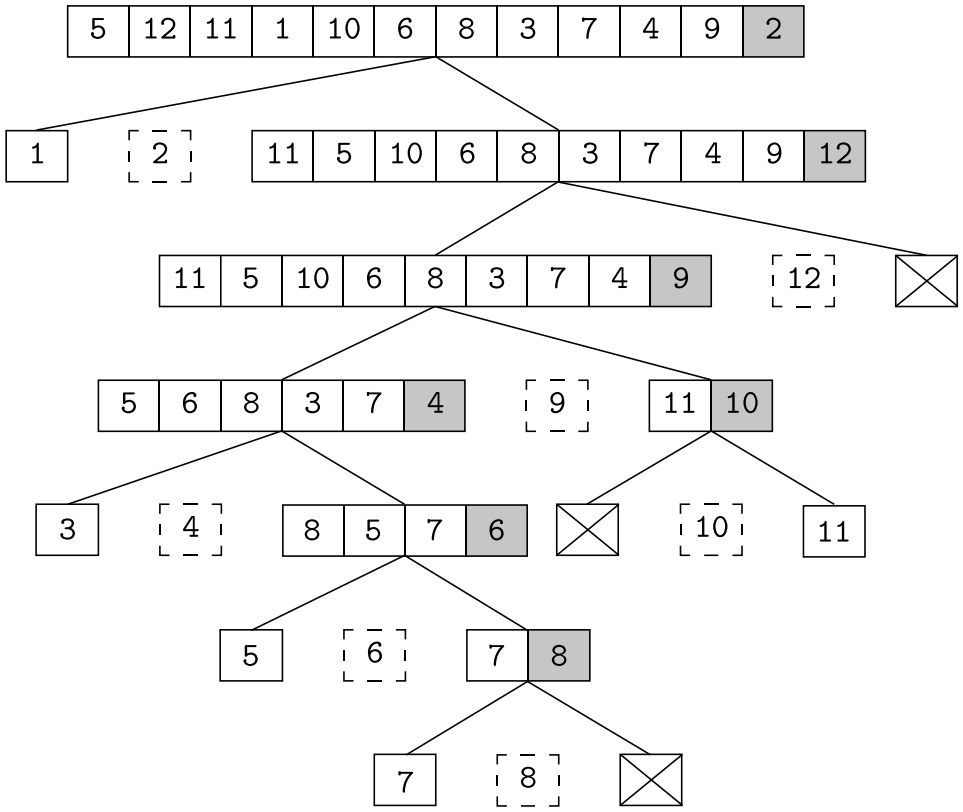



**۱)** بله. می‌توان از تجزیه Geometric نیز استفاده کرد. در این روش ابتدا با تقسیم آرایه به قسمت‌های مساوی (مثلا ۷ قسمت ۲ تایی در مثال بالا)، در هر قسمت max را در یک تسک جدید می‌یابیم. سپس هر یک از تسک‌ها با مقایسه مقدار max خودشان و global max، در صورت نیاز آن را تغییر می‌دهند. در نهایت پس از پایان عملیات همه‌ی تسک‌ها، مقدار max در global max وجود دارد.

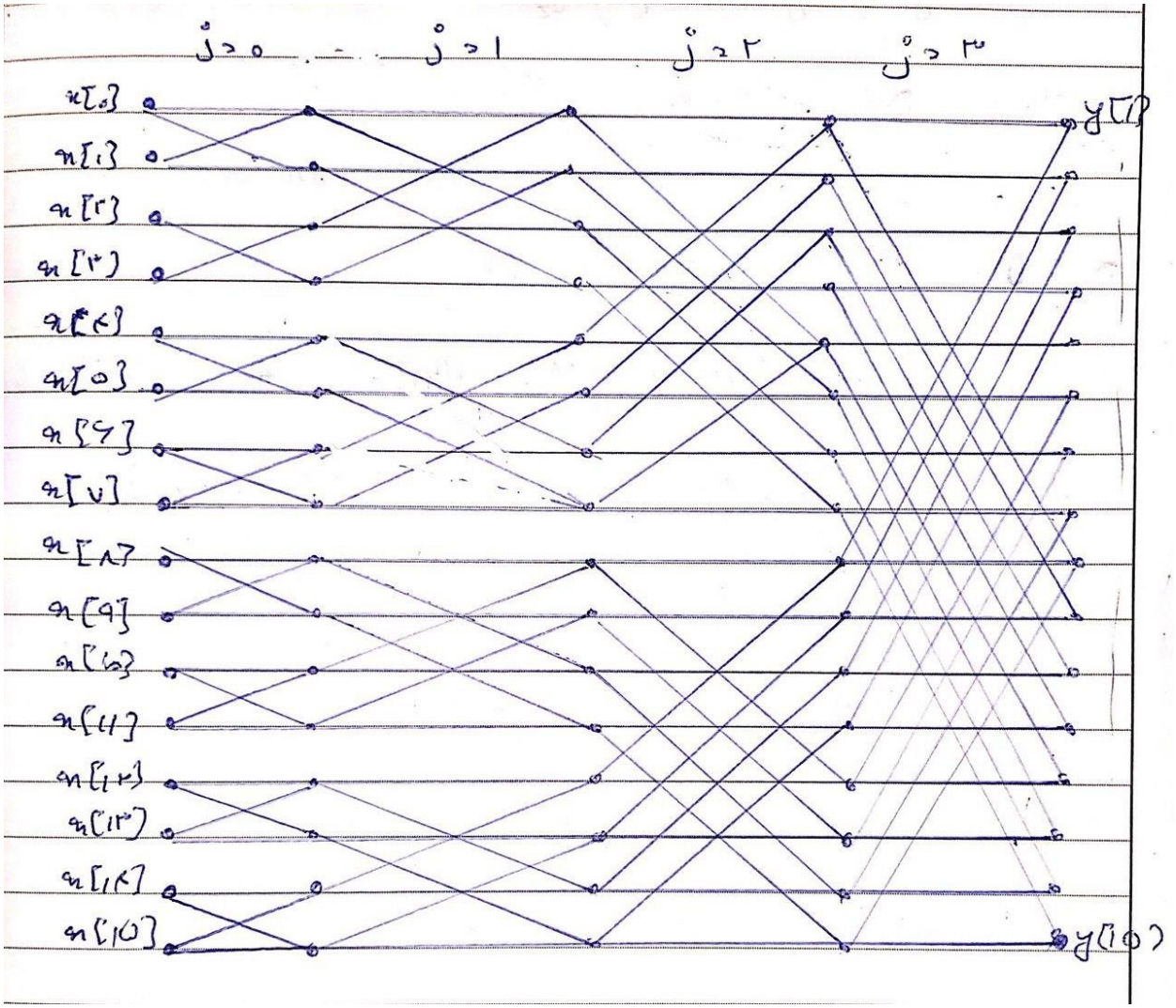
**۴.** در هر سطر، هر ستون j که مقدار صفر ندارد و j مخالف شماره سطر است، نیاز به داده (سطر jام بردار b) از تسک سطر jام دارد. در شکل زیر، اگر هر راس را یک تسک در نظر بگیریم، گراف روابط تسک‌های آن مطابق شکل زیر می‌شود.



**۵.** با در نظر گرفتن آخرین عنصر هر بخش به عنوان pivot، گراف آن به شکل زیر می‌شود. در اینجا هر تسک وظیفه partitioning آن زیربخش از آرایه را بر عهده دارد (مربع‌های با کادر نقطه‌چین برای بهتر دیده شدن عمل مرتب سازی کشیده شده اند و تسک نیستند).




**۶.** از آنجا که خط ۷ام وابستگی زیادی به حلقه بیرونی (حلقه j) دارد، حلقه داخلی (حلقه i) را باز کرده و موازی میکنیم. در نتیجه با توجه به عملگر XOR و اینکه $2^j$ تنها یک بیت آن یک است، گراف آن برای ۱۶ پردازنده به شکل زیر می‌شود:



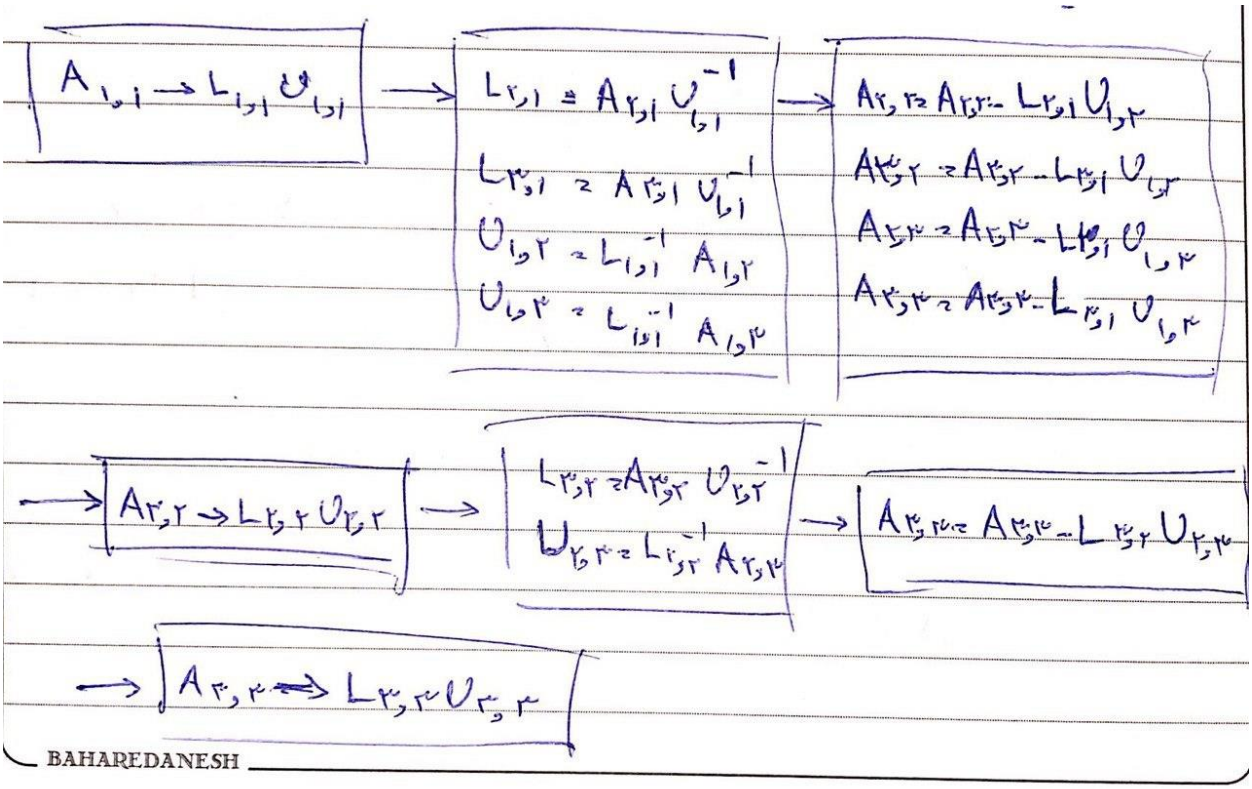
برای ۸ هسته نیز، می‌توان ۲ به ۲ تسک‌های مجاور را با هم به یک هسته نگاشت کرد.

**۷.** تجزیه speculative زمانی که برنامه ممکن است یکی از چند شاخه (branch) که از لحاظ پردازشی سنگین هستند را اجرا کند، استفاده می‌شود. در این حالت، زمانی که یک تسک در حال پردازش برای تولید خروجی که در انتخاب شاخه اجرایی پس از آن استفاده می‌شود، باقی تسک‌ها می‌توانند محاسبات هر یک از شاخه‌ها را انجام دهند. به عنوان مثال دستور switch-case در زبان C، زمانی که هنوز ورودی آن مشخص نشده است، هر یک از caseها می‌توانند به طور موازی اجرا شوند و در نهایت پس از تعیین ورودی آن، نتایج خروجی شاخه مورد نظر استفاده می‌شوند.

مزیت این روش این است که زمانی که شاخه‌ای که باید اجرا شود مشخص می‌شود، خروجی آن بلافاصله مشخص است (در صورت اتمام تسک مربوط به آن شاخه) و یا مقداری از محاسبات آن به احتمال زیاد تا به حال انجام شده است و سرعت اجرای برنامه افزایش می‌یابد.

در این روش نتایج محاسبات دیگر شاخه‌هایی که مورد استفاده قرار نمی‌گیرد باید دور ریخته شوند که عیب این روش به حساب می‌آید. در برخی کاربرد که احتمال اجرای یکی از شاخه‌ها بیشتر از دیگر شاخه‌هاست، می‌توان تنها آن شاخه را محاسبه نمود. در صورتی که نیاز به اجرای شاخه‌ی دیگری بود، محاسبات انجام شده را rollback می‌کنیم و سپس به اجرای شاخه مورد نظر می‌پردازیم. با این کار مقداری این عیب کم‌رنگ‌تر می‌شود.

**۸.** در این مسئله برای محاسبه ماتریس‌های $L$ و $U$ (که چون قطر اصلی $L$ همواره یک است، هر دو قابل ذخیره در همان یک ماتریس $A$ هستند) به ۱۴ تسک زیر نیاز داریم. تسک‌هایی که در یک دسته قرار دارند قابلیت موازی‌سازی دارند.




همان‌طور که می‌بینیم در این الگوریتم، در صورتی که به صورت بلوکی تسک‌ها را به هسته‌ها نگاشت کنیم، هسته‌هایی که خانه‌های بالا سمت چپ را محاسبه می‌کنند به مراتب کار پردازشی کمتری دارند و این باعث load-imbalance در هسته‌ها می‌شود.

با شکستن در یک بعد یا دو بعد و اختصاص دادن هر قسمت به صورت round-robin به هسته‌ها، می‌توان این پخش بار نامتوازن را متوازن کرد. به عنوان مثال در این مسئله اگر رویکرد ۲ بعدی را در پیش بگیریم، نگاشت آن به شکل زیر می‌شود:

| | | |
|---|---|---|
| P0 | P1 | P2 |
| P3 | P0 | P1 |
| P2 | P3 | P0 |