الگوریتم‌های موازی
خانم دکتر دامرودی
علیرضا سلطانی نشان
۲۸ مهر ۱۴۰۲

فهرست مطالب

۱ مفاهیم اولیه

نگارش این بخش تکمیل نشده است.

۲ وابستگی‌ها

هیچ برنامه‌ای نمی‌تواند سریع‌تر از برنامه‌ای که مدت زمان بیشتری، زمان رسیدن به نتیجه را سپری می‌کند، اجرا شود. زیرا محاسباتی که نسبت به محاسبات قبلی به یکدیگر، در زنجیره‌ای از فرایندها وابستگی دارند، باید به ترتیب اجرا شوند چرا که به صورت سریالی می‌باشند. همه الگوریتم‌ها شامل وابستگی‌هایی هستند که می‌توان آنها را تشخیص و به صورت مناسب آنها را حذف کرد تا بتوان محاسبات مستقل به صورت موازی را حل کرد.

۱.۲ انواع وابستگی‌های داده

وابستگی‌های داده به سه دسته زیر تقسیم می‌شوند:

۱. ماهیت برنامه Data Flow Dep [1]

۲. برنانویس Anti Data Dep

۳. برنامه نویس Output Data Dep

۲.۲ منبع

اگر دو دستورالعمل بخواهند به صورت همزمان کار کنند، به گونه‌ای که روی یک منبع عملیات خواندن و نوشتن را انجام دهند، دو حالت به وجود می‌آید:

۱. یا در ماهیت خود برنامه وجود دارند

۲. یا برنامه نویس ایجاد می‌کند که قابلیت حذف دارد

نکته: یکی از وظایف اصلی کامپایلر حذف وابستگی‌هایی است که برنامه نویس ایجاد می‌کند و قبل از مرحله اجرا انجام می‌شود و باعث کارایی بالا برنامه می‌شود.

وابستگی‌های زیر را در نظر بگیرید:

O .... = ... O S[i]:

O .... = ... O S[j]:

۳.۲ وابستگی WBR یا Write Before Read

این وابستگی در ابتدا روی متغیر مورد نظر داده‌ای را می‌نویسد و سپس آن را در فرایندی دیگر می‌خواند. به این نوع از وابستگی، وابستگی Date Flow گفته می‌شود.

برای مثال:

۱. C + A = K s[j] ، C + B = A s[i]:

۲. D – A = E s۲: ، C + B = A s۱:

[1] Dependency

۴.۲ وابستگی RBW یا Read Before Write

در این از وابستگی در ابتدا عملیات نوشتن روی متغیر صورت می‌گیرد، سپس از آن متغیر برای خواندن مقدار استفاده می‌کند. به این نوع از وابستگی Ani Data
برای مثال:

۱. J + K = A s[j]: ، A + C = B s[i]:

۲. ۵ – F = A s[j]: ، D * A = B s[i]:

۵.۲ وابستگی WBW یا Write Before Write

این نوع از وابستگی به شکل خطی می‌باشد. یعنی در مرحله اول فرایند ابتدا بر روی آن مقداری می‌نویسد، سپس مجددا این عمل را تکرار می‌کند. به این نوع وابستگی Output Data گفته می‌شود.
برای مثال:

۱. J + K = A s[j]: ، C + B = A s[i]:

۲. E + D = A s[۲]: ، C + B = A s[۱]:

نکته: برای نمایش وابستگی‌ها از گراف استفاده می‌کنند.
نکته: اگر تمام مسیرهای گراف به یک جهت باشد (هم جهت باشد) و حلقه وجود نداشته [۱] باشد، گراف مورد نظر به شکل آبشاری خواهد بود. که در این حالت می‌توان با اجرای مناسبی پردازشی را انجام داد.
نکته: نتیجه وابستگی‌ها به شکل جدول وابستگی‌ها یا Dependencies table نشان داده می‌شود.
[۲]

در وابسته اول، مهم‌ترین نکته در آن است که بایستی ارتباطات حداقل دو اندیس را در نظر داشته باشیم تا بتوانیم وابستگی مورد نظر را پیدا کنیم. برای مثال در این وابستگی در صورتی که تنها روی یک اندیس به دنبال وابستگی بگردیم نمی‌توانیم حلقه را پیدا کنیم. در صورتی که در ذات این برنامه حلقه یا وابستگی نوع Output date deps وجود دارد.
نکته: هر چقدر تنوع بیشتری در متغیرها وجود داشته باشد، وابستگی نیز کمتر می‌شود (فضیه گوناگونی متغیر).

---

[۱] Performancee

[۲] یادآوری: Scaler 3 of (5, 2) Vector => (15, 6)

۶.۲ انواع ایجاد وابستگی

حلقه‌ها در برنامه به دو شکل هستند:

۱. Do All: همه اندیس‌ها در دستورالعمل‌ها یکسان است.

۲. Do Across: اندیس دستورالعمل‌ها متفاوت است.

نکات:

۱. متغیرهای اسکالر باعث ایجاد وابستگی بین تکرارهای مختلف می‌شوند

۲. در حلقه Do All به دلیل یکسان بودن اندیس‌ها وابستگی ایجاد نمی‌شود.

۳. در حلقه‌های Do Across به دلیل تولید اندیس مختلف به ازای لیست‌ها، باعث ایجاد وابستگی به صورت تو در تو می‌شود. مهم ترین مثال این حلقه را می‌توان به عامل دستورالعمل A[i] = A[i + 1] / C و مانند آن اشاره کرد.

۴. وابستگی Data flow کاملا به ذات برنامه مربوط است اما وابستگی‌های Anti data و Output data به دلیل عدم رعایت اصول موازی سازی توسط برنامه نویس در برنامه ایجاد می‌شود.

۷.۲ حذف وابستگی

سه روش برای حذف وابستگی وجود دارد:

۱. Variable renaming

۲. Scaler expansion

۳. Node spliting

۱.۷.۲ حذف وابستگی با روش تغییر نام

در مثال زیر دستورالعمل‌های زیر را خواهیم داشت:

۱. A = B * C

۲. D = A + ۱

۳. A = A * D

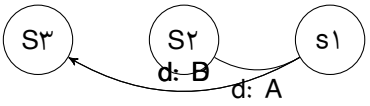
همانگونه که در صفحه پیشین گفته شد، هر چقدر تعداد متغیرهای مختلفی داشته باشیم باعث می‌شود تا وابستگی کمتری در برنامه داشته باشیم.

در این مسئله در متغیر A برای بار دوم عمل نوشتن صورت گرفته است. پس به همین خاطر بایستی دستورالعمل سوم را تغییر نام دهیم تا بتوانیم کمترین وابستگی را داشته باشیم.

۱. A = B * C

۲. D = A + ۱

۳. AA = A * D



۲.۷.۲ حذف وابستگی با روش Scaler expansion

هدف این روش حذف وابستگی‌های ناشی از متغیرهای اسکالر می‌باشد به همین دلیل یک متغییر اسکالر را به یک متغیر برداری تغییر می‌دهد.

for i = ۱ to ۱۰۰ do:

۱. b = B[i] – ۲

۲. C = C’[i] – B[i]

۳. A[i] = b + C

حل:

زمانی که حلقه در i=1 است:

for i = ۱ to ۱۰۰ do:

۱. b = B[۱] – ۲

۲. C = C’[۱] – B[۱]

۳. A[۱] = b + C

زمانی که حلقه در i=2 است:

for i = ۲ to ۱۰۰ do:

۱. b = B[۲] – ۲

۲. C = C’[۲] – B[۲]

۳. A[۲] = b + C

دلیل اصلی تبدیل متغیر اسکالر به متغیر برداری (از نوع لیست با اندیس) آن است، در صورتی که از متغیر اسکالر استفاده شود در حقیقت مرجعی در تمامی فرایندهای حلقه وجود دارد. برای مثال توجه شود که در اندیس ۱ که حلقه شروع می‌شود سه دستورالعمل انجام شده و زمانی که وارد حلقه دوم با اندیس ۲ شود متغیرهای b و C در حقیقت رفرنس حلقه‌بالایی هستند که می‌توانند به نحوی انجام شوند که موجوب تولید output data Dependency و anti data Dependency شود. به همین جهت می‌توان دو وابستگی مذکور که طی فرایند حاصل شده است را با این روش به سادگی از بین برد.

for i = ۱ to ۱۰۰ do:

۱. b[i] = B[i] – ۲

۲. C[i] = C'[i] – B[i]

۳. A[i] = b[i] + C[i]

حل:

زمانی که حلقه در i=1 است:

for i = ۱ to ۱۰۰ do:

۱. b[۱] = B[۱] – ۲

۲. C[۱] = C'[۱] – B[۱]

۳. A[۱] = b[۱] + C[۱]

زمانی که حلقه در i=2 است:

for i = ۲ to ۱۰۰ do:

۱. b[۲] = B[۲] – ۲

۲. C[۲] = C'[۲] – B[۲]

۳. A[۲] = b[۲] + C[۲]

۳.۷.۲ حذف وابستگی با استفاده از روش Node splitting

وابستگی بین متغیرهای برداری حلقه‌های Do across را حذف می‌کند. برای این منظور متغیر برداری را با یک متغیر برداری دیگر تغییر نام یا rename می‌کند تا اندیس‌های آنها یکسان شود و تبدیل به یک حلقه Do across شود.

مثال:

for i = ۱ to ۱۰۰ do:

۱. A[i] = B[i] − C[i]

۲. D[i] = A[i] + ۲

۳. F[i] = D[i] + $\mathrm{A[i+1]}$

حل:

for i = ۱ to ۱۰۰ do:

۱. AA[i] = $\mathrm{A[i+1]}$

۲. A[i] = B[i] − C[i]

۳. D[i] = A[i] + ۲

۴. F[i] = D[i] + AA[i]

مثال:

وابستگی در مسئله زیر پیدا کرده و گراف آن را بکشید

for i = ۱ to ۱۰۰ do:

۱. A[i] = C + D[i]

۲. B[i] = $\mathrm{A[i+1]}$ + K

۳. K = C + T

در این مسئله با وجود $\mathrm{A[i+1]}$ می‌توان گفت حذف نوع سوم باید صورت گیرد. به دلیل آن که K به صورت اسکالر می‌باشد و می‌تواند بارها تکرار شود و وابستگی نوع output را بسازد باید از حالت اسکالر به بردار تبدیل شود که در اندیس‌های متفاوت از حافظه به خواندن و نوشتن بپردازد.

for i = ۱ to ۱۰۰ do:

۱. AA[i] = $\mathrm{A[i+1]}$

۲. A[i] = C + D[i]

۳. B[i] = AA[i] + K[i]

۴. K[i] = C + T

۳ توازی در statement