گزارش طرح تحقیقاتی

موازی سازی اتوماتیک برنامه های ترتیبی

تهیه کننده: سعید پارسا

بهار سال
۱۳۸۶

بنام خداوند

بخشنده مهربان

چکیده

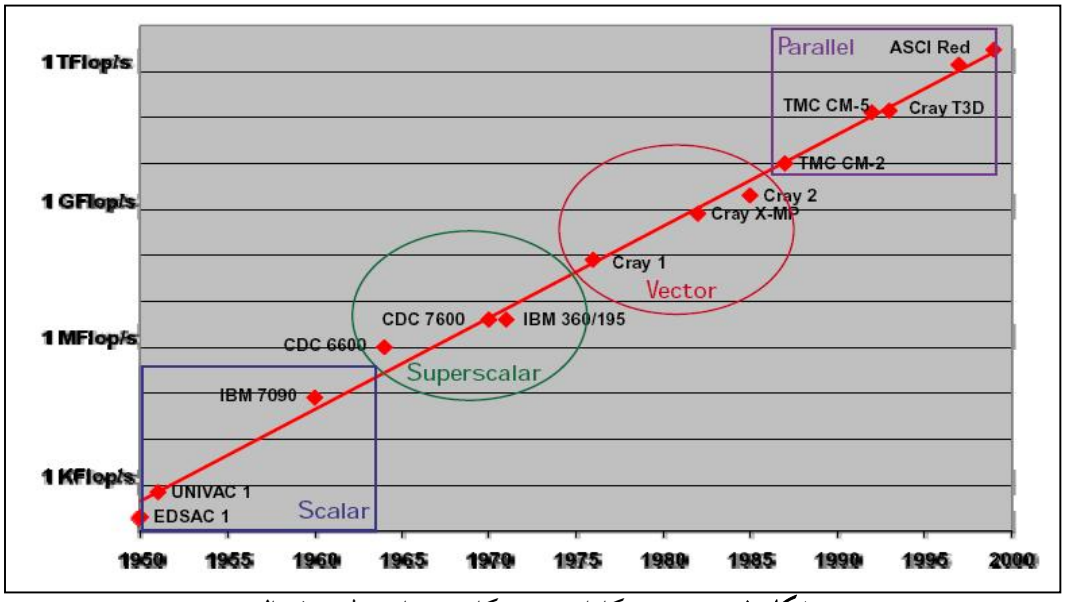
در این طرح تحقیقاتی مراحل تبدیل اتوماتیک کد ترتیبی به کد موازی مشخص شده است. وابستگی بین جملات مغایر با اجرای موازی بخشهای در برگیرنده جملات وابسته با یکدیگر است. وابستگیها در قالب دو نوع وابستگی داده ای و کنترلی قابل تشخیص و در قالب گراف وظایف مدل می شود. مساله دانه بندی وظایف و زمانبندی آنها برای اجرای موازی است. باید بتوان وظایف را آنچنان در بین پردازنده ها توزیع نمود که کلیه پردازنده ها بطور همزمان فعال و با حداقل نیاز به نتایج محاسبات در سایر پردازنده ها فعال باشند. در این راستا مقوله های زمانبندی گراف وظایف بسیار مطرح می باشد.

نوع دیگر وابستگی در بین تکرارهای حلقه ها مطرح است. وابستگی بین تکرارهای حلقه می تواند منجر به حل دستگاه نا معادلات خطی و غیر خطی شود. به علت زیاد بودن تعداد وابستگیها می توان مجموعه پایه از بردارهای وابستگی در بین تکرارهای حلقه را مشخص نمود. به این ترتیب فضای یکنواخت وابستگی بین تکرارهای حلقه حاصل می گردد. دانه بندی کاشی ها و تعیین کاشی هایی که به صورت موازی قابل اجرا هستند مساله دیگری است که می بایست مورد بررسی قرار گیرد. مساله زمانبندی و توزیع تکرارهای حلقه در بین پردازنده ها خود نکته قابل توجه دیگری می باشد.

فهرست مطالب

۱. سوپر کامپیوترها

بر طبق قانون Moore در هر ۱۰ سال توان عملیاتی سوپر کامپیوترها دو برابر می شود. در شکل ذیل برای نمونه روند افزایش کارایی سوپر کامپیوترها در طی ۵۰ سال مشخص شده است.

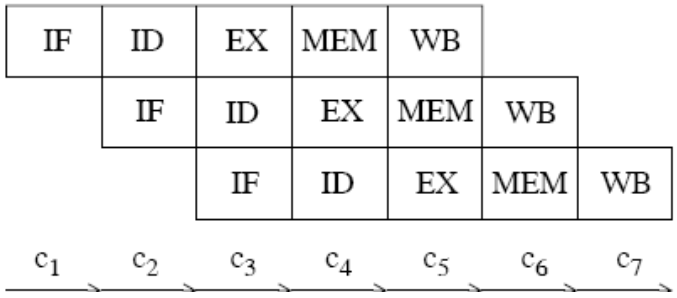


شکل ۱– روند رشد کارایی سوپر کامپیوترها در طی ۵۰ سال

البته امروزه با پیشرفت فنآوری‌هایی از قبیل پردازش تصویر و سیستم‌های چند رسانه ای نیاز بیشتری به سیستم‌ها با توان پردازش و کارایی بالا بوجود آمده است. به موازات افزایش سرعت اجرایی پردازنده ها، می بایست زمان لازم برای دسترسی نیز کاهش داده شود. برای این منظور حافظه های نهان در چند سطح دسترسی مطرح شده اند. با افزایش توان پردازشی، کار برنامه سازی برای سیستم‌های موازی نیز مشکلتر می شود. برنامه نویس می بایست بطور دستی برنامه های خود را تبدیل به کد موازی نمایند. فلسفه وجود کامپایلرهای موازی کننده به حداقل رساندن میزان تبدیل‌های دستی است.

۲. خط لوله

یکی از کاربردهای موازی سازی برای افزایش کارایی، بکارگیری خط لوله می باشد. برای نمونه در شکل ذیل خط لولـه در ماشینهای IBM 7094 مشخص شده است. در این نوع کامپیوترها هر دستورالعملی در دو مرحله انجام می شد. مرحلـه اول در ارتباط با بار کردن آوردن دستورالعمل از حافظه به درون پردازنده و مرحلـه دوم اجـرا کـردن دسـتورالعمل مـی باشـد. چـون دستورالعملها را می توان از دیتا بانکهای مختلف حافظه بطور همزمان دسترسی نمود، می توان ایـن دو مرحلـه آوردن و اجـرا کردن را با یکدیگر همگام نمود. بعد از آن در IBM 370 خط لوله به صورت چهار مرحله ای تبدیل شد.



شکل ۱- نمونه ای از خط لوله در ماشینهای ریسک

خط لوله شامل مراحل ذیل است:

1. instruction fetch (IF),
2. instruction decode (ID)
3. execute (EX) and
4. memory access (MEM) and
5. write back (WB)

در ماشینهای ریسک دستورالعملها به سه دسته تقسیم می شوند:

۱- عملیات ثبات به ثبات واحد محاسباتی: اینها شامل عملیاتی محاسباتی مثل جمع و تفریق می باشند.

۲-عملیات بر روی حافظه : نیاز به دسترسی به حافظه در ضمن اجرای عملیات.

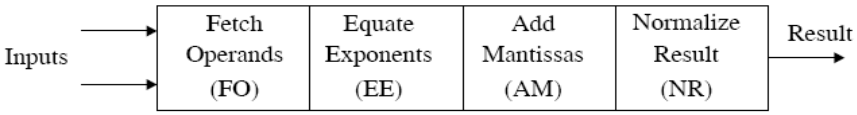
۳- عملیات پرش : بر اساس یک شرط موجب تغییر آدرس برای آوردن دستورالعمل بعدی می شوند.

چنانچـه دسـتورالعمل از نـوع اول باشـد و در داخـل واحـد محاسـبات انجـام گیـرد بـا مرحلـه اجـرا خاتمـه مـی یابـد. چنانچـه دستورالعمل نیاز به دسترسی به حافظه داشته باشد، دسترسی به حافظه در مرحله MEM انجام می گیرد. چنانچه داده مورد نظر درون حافظه نهان نباشد، در این صورت اجرای دستورالعمل تا آوردن داده از حافظه متوقف مـی شـود. چنانچـه دسـتورالعمل پرش شرطی باشد، در مرحله اجرا یا EX ثبات مشخص شده در دستورالعمل پرش با صفر مقایسه می شود. در صـورت صـفر

بودن، مقدار شمارنده آدرس مساوی با آدرس مشخص شده در دستورالعمل می گردد. مرحله WB یا Write Backبرای بازنویسی مقادیر در ثباتها است.

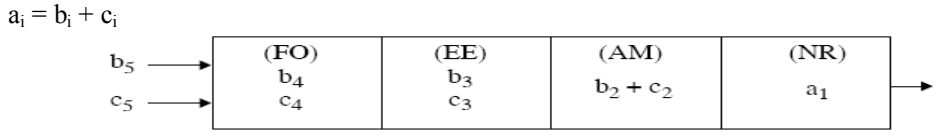
۳. واحدهای اجرایی خط لوله

اغلب دستورالعملها بیش از یک سیکل اجرایی را لازم دارند. برای نمونه محاسبات اعشاری معمولاً چهار یا پنج سیکل اجرایی را نیاز دارند. در جمع اعشاری پس از اینکه دو عملوند از حافظه دستیابی شدند، می بایست توانهای آنها یکسان یا در اصطلاح نرمالیزه شود تا بتوان مانتیسها را با یکدیگر در مرحله بعدی جمع نمود. بالاخره نتیجه می بایست قبل از اینکه ذخیره شود نرمالیزه شود.



شکل ۲- نمونه ای از جمع کننده اعشاری

از آنجاییکه هر بخش از عملیات در واحد جمع مستقل از سایرین است، هر بخش می تواند بر روی عملوندهایی متفاوت از سایرین عمل نماید. به این ترتیب چنانچه نیاز به محاسبه چند حاصل جمع بطور همزمان باشد، می توان محاسبات را با همزمان نمودن بخشها به صورت ذیل تسریع نمود.



شکل ۳- نمونه ای از خط لوله برای محاسبه $a_i = b_i + c_i$

چنانچه زمان لازم برای انجام هر مرحله یا بخش از خط لوله یک سیکل باشد، آنگاه زمان لازم برای انجام n عمل جمع اعشاری 4n است. چنانچه بخشها همپوشانی داشته باشند، آنگاه زمان لازم به n+3 سیکل کاهش داده می شود. یک واحد خط لوله هنگامی موثر است که خط لوله پر باشد. بنابراین برای استفاده موثر از واحدهای خط لوله نیاز است که کامپایلر به قسمی دستورالعملها را مرتب نماید که عملوندها به موقع در داخل خط لوله قرار گیرند تا خط لوله پر باقی بماند.

۳. واحدهای اجرایی خط لوله

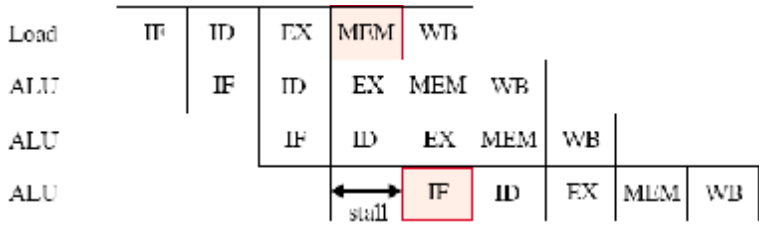
در معماری‌های خط لوله بزرگترین مانع وجود موانع یا در اصطلاح stall های خط لوله است. مانع هنگامی رخ می دهد که مجموعه جدید ورودی را نتوان در خط لوله بواسطه شرایطی به نام مخاطره یا hazard به اجرا در آورد. هنسی و پاترسون مخاطرات را به سه دسته تقسیم می کنند:

۱. مخاطرات ساختاری، بدین واسطه رخ می دهند که منابع ماشین نتوانند کلیه ترکیبهای همپوشانی دستورالعمل را پشتیبانی کنند.

۲. مخاطرات داده ای، هنگامی که نتیجه حاصل از یک دستورالعمل مورد نیاز دستورالعمل بعدی باشد.

۳. مخاطرات کنترلی، هنگام پردازش دستورالعملهای انشعاب ایجاد می شود.

مخاطره ساختاری میتواند در صورت عدم وجود منبع رخ دهد. برای نمونه اگر ماشینی صرفاً یک درگاه برای حافظه داشته باشد و نتوان داده و دستورالعمل را به طور همزمان از حافظه خواند این مشکل می تواند رخ دهد. برای نمونه به شکل ذیل توجه نمایید:



شکل ۴- مخاطره ساختاری در درگاه حافظه ماشینهای DLX

همانگونه که در بالا مشخص است نمی توان چهارمین fetch از حافظه را انجام داد زیرا دستورالعمل اولی در این زمان نیاز به دسترسی به حافظه دارد. برای ممانعت از مواجه شدن با همچنین وضعیتی در هنگام تولید کد سعی به استفاده بیشتر از ثباتها می شود. برای نمونه به دستورالعملهای ذیل توجه نمایید:

ADD R1,R2,R3
SUB R4,R1,R5

این دو دستورالعمل بدون هیچگونه تاخیری قابل اجرا هستند زیرا نتیجه دستورالعمل اول در دومی بلافاصله استفاده می شود. اما، در مورد دو دستورالعمل زیر نمی توان از ایجاد مانع یا در اصطلاح stall جلوگیری نمود.

LW R1,0(R2)

ADD R3,R1,R4

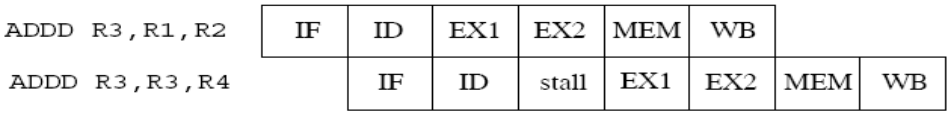
علت بوجود آمدن مانع در شکل ذیل مشخص شده است:

LW R1,0(R2)	IF	ID	EX	MEM	WB		
ADD R3,R1,R4		IF	ID	stall	EX	MEM	WB

شکل ۴- مخاطره داده ای

می توان با استفاده از امکان زمانبندی دستورالعملها یا در اصطلاح Instruction Scheduling یا Instruction Reordering این مشکل را حل نمود. برای این منظور با در نظر گرفتن وابستگی های داده ای و کنترلی امکان جابجایی دستورالعملهایی که ثبات R1 را استفاده نمی کنند و مشکل stall را ایجاد نمی کنند، را به میان دو دستورالعمل فوق بررسی باید نمود. در صورت یافتن دستورالعمل مناسب آنرا به بین دو دستورالعمل می توان انتقال داد.

مشکل فوق در اجرای عبارتی مثل A+B+C نیز امکان پذیر است. برای نمونه به شکل ذیل توجه نمایید. در اینجا چنانچه اجرا از چپ به راست انجام شود دستورالعمل جمع دومی می بایست برای یک سیکل در انتضار اجرا باقی بماند.



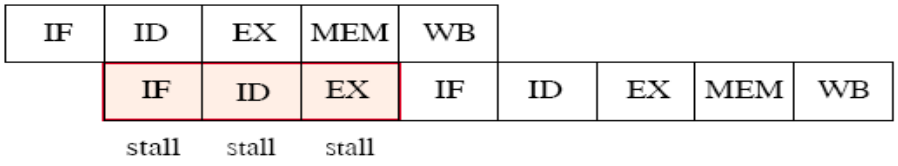
شکل ۵- مخاطره داده ای ایجاد کننده مانع

بازهم می توان گفت که زمانبند کامپایلر می تواند مانع این مشکل شود. برای نمونه فرض کنید که عبارتی برای جمع چهار متغیر به صورت A+B+C+D قرار است اجرا شود. در اینجا بعد از هر عمل جمع به تعداد دو سیکل انتظار برای دریافت نتیجه دستورالعمل قبلی نیاز است. برای رفع مشکل کامپایلر می تواند عمل جمع را به صورت ذیل گروه بندی کند:

(A+B) + (C+D)

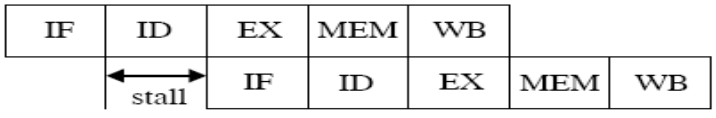
در این صورت جمع دومی می توان در یک سیکل بعد از جمع اولی به اجرا در آید. در مجموع یک سیکل مجموع سریعتر بدست می آید.

مخاطرات کنترلی همانطور که در بالا توضیح داده شد بواسطه دستورالعملهای پرش ایجادمی شوند. در ماشینهای DLX مخاطرات کنترلی می تواند مانعی به طول سه سیکل زمانی ایجاد نماید. پردازنده شروع به آوردن دستورالعمل بعدی از حافظه می کند بدون اطلاع از اینکه پرش انجام می شود. تشخیص عمل پرش بعد از مرحله MEM از دستورالعمل پرش مشخص می شود. در این صورت دستورالعمل بعدی می بایست از درون حافظه بیرون کشیده شود قبل از اینکه هر گونه تغییری در ثباتها و یا حافظه ایجاد شود. برای نمونه به شکل ذیل توجه نمایید.



شکل ۵- مخاطره ایجاد شده بواسطه دستورالعمل پرش شرطی

از آنجاییکه سه سیکل پنالتی زیادی می باشد، طراحان ماشین سعی زیادی به کاهش این هزینه نموده اند. یک راه، استفاده از سخت افزاری است که در ضمن دیکد دستورالعمل بتواند تشخیص پرش را بدهد. این عمل در صورت آزمون مقایسه با صفر امکان پذیر است. در این صورت stall به یک عدد به صورت زیر تقلیل داده می شود:



شکل ۶- تقلیل مخاطره ایجاد شده بواسطه دستورالعمل پرش شرطی

۳. پردازنده‌های برداری

در سال ۱۹۷۰ وظیفه پر نگهداشتن خط لوله مساله ای آزار دهنده شده بود. لذا، در سوپر کامپیوترها طرح سخت افزاری را مطرح نمودند که بتوان برای اجرای بدون وقفه دنباله ای از دستورالعملها به جلو نگاه کرد. برای این منظور دستورالعملهای برداری یا vector instructions مطرح شدند. در این نوع ماشینها با اجرای یک دستور العمل بر روی دو آرایه از ثباتها یا درون حافظه می توان عمل نمود. مقادیر را به درون آرایه ثباتها نیز با یک دستورالعمل می توان انتقال داد. برای نمونه ماشین Cray 1 که در سال ۱۹۷۵ ایجاد شده دارای ۷ آرایه ثبات ۶۴ خانه ای است. امکان ایجاد زنجیره ای از عملیات برداری نیز در این نوع کامپیوترها بوجود آمد. عمل زنجیر کردن در واقع بواسطه یک عمل برداری که در انتظار دریافت خروجی از عمل برداری دیگر است انجام می شود. برای نمونه در ادامه عمل جمع دو آرایه ۶۴ خانه ای A و B و قرار دادن حاصل در آرایه C مشخص شده است:

```
VLOAD V1,A
VLOAD V2,B
VADD V3,V1,V2
VSTORE C,V3
```

دستورالعمل جمع فوق را می توان به صورت ذیل در فرترن ۹۰ مشخص نمود:

```
C(1 : 64) = A(1 : 64) + B(1 : 64)
```

مساله تبدیل خودکار دستورالعملها به فرم برداری است. برای نمونه حلقه ذیل را در نظر بگیرید:

```
For i:= 1 to 4 do c[i] := a[i]*B[i]
```

در اینجا مشکل در تبدیل به فرم برداری است. در پردازنده برداری قبل از اجرا مقادیر می بایست از حافظه خوانده شده باشند یا در اصطلاح load شده باشند. در حالیکه در داخل حلقه مقادیر یک به یک خوانده می شوند. بعضا" این امر مشکلساز است. لذا، مساله قابل تبدیل بودن به فرم برداری یا vectorizable بودن کد مطرح شد. برای نمونه حلقه ذیل قابل بردار سازی نیست:

```
For i:= 1 to 4 do c[i] := a[i]*B[i]
```

در اینجا نتیجه هر تکرار در تکرار بعدی استفاده می شود بنابراین امکان اجرای حلقه به صورت موازی وجود ندارد. اما امکان اجرای دستورالعمل ذیل در ماشین برداری وجود دارد:

```
A(2:65) = A(1:64) + B(1:64)
```

۴. پردازنده‌های سوپر اسکالر و پردازنده های VLIW

مشکل پردازنده های برداری مجموعه نسبتاً بزرگ دستورالعملها است. در این دسته از پردازنده‌ها علاوه بر دستورالعملهای عادی می بایست مجموعه دستورالعملهایی برای انجام عملیات برداری وجود داشته باشد. برای رفع مشکل سعی شد در ماشینهای تک پردازنده ای سرعت اجرایی را به حد پردازنده های برداری افزایش دهند. اگر بتوان یک یا چند دستورالعمل خط لوله را بطور همزمان به اجرا در آورد آنگاه شاید بتوان سرعت اجرایی پردازنده های برداری را حاصل نمود. این ایده

اولیه ایجاد پردازنده های سوپر اسکالر و VLIW بوده است. در این نوع ماشینها بیش از یک دستورالعمل را می شد در هر سیکل به اجرا در آورد.

در ماشینهای سوپراسکالر اجرای همزمان دستورالعملها به این صورت رخ می دهد که پردازنده به جلو می نگرد تا دستورالعملهایی که آماده اجرا هستند را مشخص نماید. از بین دستورالعملها به ترتیب آنهایی را که قابل اجرا شدن هستند را به اجرا در می آورد. در برخی از ماشینها حتی ترتیب اجرایی رعایت نمی شود.

در پردازنده های VLIW می توانند چندین دستورالعمل را با اجرای یک دستورالعمل بزرگ یا پهن به اجرا در درآورند. یک دستورالعمل پهن خود دارای چندین دستورالعمل کوچک است که بطور همزمان به اجرا در می آیند. بنابراین اگر یک ماشین VLIW دارای دو واحد خط لوله ای ضرب ممیز شناور است، آنگاه می تواند دو عمل ضرب را بطور همزمان و در یک سیکل به اجرا در آورد. در ماشینهای VLIW می بایست کامپایلر و یا برنامه نویس دستورالعملها را در قالب دستورالعملهای پهن دسته بندی کنند. در این دسته ها می بایست ورودی و خروجی دستورالعملها آماده باشد.

مشکلات پردازنده های سوپراسکالر و VLIW اولاً نیاز مبرم آنها به پهنای باند بیشتر برای آوردن دستورالعملها از حافظه است. برای این منظور نیاز به حافظه های پنهان نسبتاً بزرگ برای نگهداری دنباله دستورالعملها است. به همین ترتیب برای نگهداری عملوندها نیاز به حافظه های پنهان بسیار بزرگ است. بطور خلاصه می توان پردازنده های سوپر اسکالر و vliw را مشبه پردازنده های برداری دانست زیرا موازیسازی مبتنی بر بردارربندی را پشتیبانی می‌کنند. تعداد زیادی از ریزپردازنده های جدید مثل Motorola G4 که در اپل مکین تاش رایج می باشند دارای comprocessor هایی برای عملیات برداری هستند که در اصل vliw می باشند.

۴. کامپایل برای سوپر اسکالر و پردازنده های VLIW

می بایست کامپایلرها با تشخیص وابستگی و زمانبندی صحیح دستورالعملها، امکان اجرای هر چه سریعتر برنامه ها را مهیا نمایند. البته با استراتژیهای نگرش به دستورالعملهای بعدی در برخی از پردازنده های سوپراسکالر هیچگونه نیازی به زمانبندی دستورالعملها نیست. برای زمانبندی دستورالعملها هر پردازنده می بایست بداند که کدام دستورالعملها وابسته به دیگری هستند و برای هر وابستگی چقدر تاخیر زمانی بین دو دستورالعمل لازم است. برای نمونه به دنباله دستورالعملهای ذیل توجه نمایید:

```
LD R1,A
LD R2,B
FADD R3,R1,R2
STD X,R3
LD R4,C
FADD R5,R3,R4
STD Y,R5
```

با توجه به اینکه هر دستورالعمل در یک سیکل آماده اجرا می شود و نتیجه بارکردن یا load با دو سیکل تاخیر در اختیار قرار می گیرد، و همچنین برای جمع ممیز شناور دو سیکل تاخیر وجود دارد، می توان با قرار دادن دستورالعمل لود برای C قبل از استور X مدت زمان اجرایی را کاهش داد.

```
LD R1,A
LD R2,B
LD R4,C
FADD R3,R1,R2
FADD R5,R3,R4
STD X,R3
STD Y,R5
```

اولین زمانبندی ۱۱ سیکل زمانی طول می کشد. ۷ سیکل برای اجرای دستورالعملها بطور عادی و ۴ سیکل تاخیر بخاطر دستورالعملهای جمع و استور، ایجاد می شود. زمانبندی دومی تنها ۸ سیکل زمان احتیاج دارد. زیرا تنها تاخیر مورد نیاز در بین دو دستورالعمل جمع است. مجموعه دستورالعملهای فوق برروی ماشینی که قابلیت اجرای بیش از یک دستورالعمل در واحد زمانی را دارد قابل تسریع نیست. اما دنباله دستورالعملهای ذیل که دو عمل جمع مستقل را انجام می دهند بر روی ماشین VLIW قابل بهینه سازی است:

```
LD      R1,A
LD      R2,B
FADD    R3,R1,R2
STD     X,R3
LD      R4,C
LD      R5,D
FADD    R6,R4,R5
STD     Y,R6
```

در یک ماشین VLIW با قابلیت اجرای ۲ جمع و ۲ لود در یک سیکل، می توان بطور کامل محاسبه دومین عبارت را با عبارت اول همپوشانی داد.

```
LD      R1,A        LD R4,C
LD      R2,B        LD R5,D
delay               delay
FADD    R3,R1,R2    dela:
STD     X,R3        FADD R6,R4,R5
empty               STD Y,R6
```

در حالیکه خط لوله راه کار موثری برای افزایش سرعت اجرایی یک پردازنده می باشد، موازی سازی پردازنده ها با اجرای بخشهای مختلف یک برنامه بر روی چند پردازنده و یا اجرای چند برنامه بطور همزمان بر روی چند پردازنده امکان کاهش زمان اجرایی را فراهم نموده است. دو روش موازی سازی همزمان و غیر همزمان رایج می باشد. در موازی سازی همزمان کپی های مختلف یک برنامه بطور همزمان بر روی داده های مختلف به اجرا در می آیند. این نوع پردازنده ها در اجرای کد حاوی پرش های شرطی موفق نیستند زیرا وابسته به اینکه کدام شاخه اجرا شود پردازنده های مربوط به آن شاخه فعال شده و پردازنده هایی که برای اجرای شاخه دوم تخصیص شده اند در این ضمن غیر فعال می شوند.

در موازی سازی غیر همزمان پردازنده ها به هر پردازنده این امکان داده می شود که برنامه ای متفاوت یا بخشی از یک برنامه متفاوت از سایرین را به اجرا در آورد. ارتباط بین این نوع از پردازنده ها ممکن است به صورت مستقیم و یا از طریق حافظه مشترک برقرار گردد. به این نوع از پردازنده ها چند پردازنده ای های متقارن[1] اتلاق می شود. سازنده هایی مثل

[1] Symetric Multi-Processors (SMP)

compaq، HP، IBM، اینتل، Sequenty و سیلیکون گرافیک این نوع پردازنده ها را ارایه می دهند. در این نوع ماشینها همزمانی بین پردازنده ها در صورت نیاز می بایست صریحاً مشخص شود.

اصولا" مشکل اساسی در دسترسی به حافظه با افزایش نسبی سرعت پردازنده ها بوجود آمده است. بخصوص این مساله با افزایش اندازه حافظه بیشتر نمایان می گردد. دو معیار برای اندازه گیری کارایی سیستم حافظه مطرح می باشد:

۱- latency یا تاخیر زمانی که نمایانگر تعداد سیکلهایی که برای انتقال داده از حافظه مورد نیاز است می باشد.

۲- Bandwidth یا پهنای باند که تعداد عناصر داده ای مورد نیاز برای انتقال از پردازنده به حافظه را در هر سیکل مشخص می کند.

به عبارت دیگر Latency شاخص مدت زمانی است که پردازنده می بایست در انتظار برای دریافت داده مورد نظر از حافظه باشد. پهنای باند مشخص می کند چه تعدادی عملیات بر روی حافشه در هر سیکل را می توان پشتیبانی نمود. با افزایش پهنای باند مقادیر بیشتری را می توان بطور همزمان از حافظه دریافت نمود. برای رفع مشکل از ساختار سلسله مراتبی حافظه استفاده شده است. به این ترتیب داده ها و دستورالعملهای تکراری در حافظه سریعتر انتقال داده می شود.

۵. موازی سازی در سطح دستورالعمل‌ها

می توان با استفاده از معماری خط لوله توازی را در اجرای بخشهای متفاوت دستورالعملهاس اسمبلی ایجاد نمود. در شکل ذیل یک خط لوله پنج مرحله ای ارایه شده است. چنانچه نتیجه یک دستورالعمل تا قبل از اجرای دستورالعمل بعدی آماده باشد پردازنده می تواند در هر کلاک یک دستورالعمل را به اجرا درآورد. در شکل ذیل یک نمونه از خط لوله پنج مرحله ای برای اجرای ۵ دستو.رالعمل متوالی مشخص شده است. ۵ مرحله شامل:

۱- آوردن دستورالعمل Instruction Fetch (IF)،

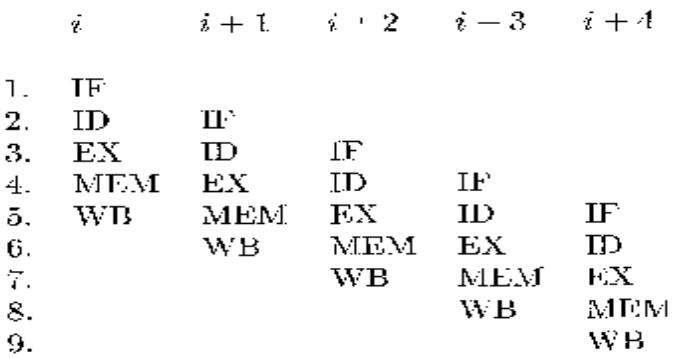
۲- رمزکشایی دستورالعمل Instruction Decode (ID)

۳- اجرای دستورالعمل Execute (EX)

۴- دسترسی به حافظه Memory Access (Mem)

۵- بازنویسی نتایج Write Back the results (WB)

است.



شکل ۵-۱ اجرای ۵ دستورالعمل در خط لوله ۵ مرحله ای

با اجرای یک دستورالعمل پرش خط لوله خالی می شود. تا دستورالعملهای بعدی fetch شوند. این تاخیر برای آوردن دستورالعمل بعدی موجب می شود تا سکسکه یا در اصطلاح hiccups در خط لوله دستورالعملها ایجاد شود. در پردازنده های پیشرفته بر اساس سابقه اجرایی سعی می شود تا مقصد دستورالعملهای پرش مشخص و از قبل به داخل خط لوله آورده شود.

۵-۱ اجرا در خط لوله (ص ۷۳۳ بخش ۱۰,۱,۳)

برخی از دستورالعملها مثل آوردن مقدار از حافظه چند کلاک طول می کشند. بنا بر تعریف در صورتی یک دستورالعمل در خط لوله قابل اجرا است که دستورالعملهای بعدی در صورت عدم وابستگی آنها به نتایج دستورالعمل کنونی بتوانند به اجرای خود ادامه دهند. بنابراین در صورتی یک خط لوله n مرحله ای باشد، حداکثر n دستورالعمل بطور همزمان قابل اجرا خواهند بود. اغلب سخت افزارهای خط لوله وابستگی بین دستورالعملهای متوالی را تشخیص داده و بطور خودکار اجرای دستورالعمل را در صورت آماده نبودن عملوندها متوقف یا در اصطلاح stall می کنند.

در برخی از سخت افزارها با افزایش تعداد واحدهای عملیاتی امکان اجرای تا m دستورالعمل بطور همزمان در خط لوله n مرحله ای فراهم گشته است. بدین ترتیب امکان اجرای همزمان حداکثر تا m*n دستورالعمل فراهم شده است. موازی سازی چنانچه در سطح نرم افزار برای این نوع سخت افزارها صرفاً امکان پذیر باشد، آنرا VLIW نامند. در غیر این صورت چنانچه موازی سازی در سطح سخت افزار انجام گیرد، آنرا سوپر اسکالر یا Super-Scaler می نامند. سوپر اسکالرها قابلیت تشخیص وابستگی در بین دستورالعملها را دارند و آنها را در صورت آماده بودن عملوندها به احرا در می آورند. این نوع ماشینها دارای یک زمانبند ایستا هستند که دستورالعملهای غیر وابسته را بر اساس ترتیب اجرایی در کنار یکدیگر قرار می دهد.

جهت ممانعت از وقفه های حاصل از عدم وجود داده مورد نیاز در حافظه اصلی و یا ثبات مورد نظر در برخی سخت افزارها دستورالعمل prefetch امکان می دهد تا به پردازنده تفهیم شود که مقداری از حافظه به حافظه نهان انتقال داده شود تا در آینده نزدیک استفاده شود. با استفاده از بیت اضافی در ثباتها به نام Poison Bit در صورت آماده نبودن داده مورد نظر وقفه ایجاد نمی شود صرفاً بیت poison ثبات مقصد یک می شود.

از آنجاییکه دستورالعملهای پرش بخصوص در خط لوله مشکل ساز و موجب سکسکه یا stall می شوند، سعی شده تا تداد دستورالعملهای پرش به حداقل ممکن کاهش داده شود. برای نمونه دستورالعمل انتقال شرطی در ادامه ارایه شده است.

دستورالعمل CMOVZ R2,R3,R1 این مفهوم را دارد که محتوی ثبات R3 به R2 انتقال داده می شود. به این ترتیب دستورالعمل :

If (a == 0) b = c + d;

با فرض اینکه مقدار a, b, c و d در ثباتهای به ترتیب R1, R2, R4 و R5 هستند، دستورالعمل ذیل ایجاد می شود.

ADD R3, R4, R5
CMOVZ R2, R3, R

به این ترتیب وابستگیهای کنترلی را می توان تبدیل به وابستگیهای داده ای نموده، اندازه بلاکهای اولیه را افزایش داد.

۵-۲ زمانبندی دستورالعملها (ص۷۴۵ کتاب اهو)

برای اجرای بدون وقفه در خط لوله دستورالعمل‌ها را می توان با در نظر گرفتن ترتیب وابستگی بین آنها جابجا نمود. به قسمی که حتی المقدور و تا حد ممکن دستورالعمل های غیر وابسته در کنار یکدیگر قرار بگیرند. جابجایی در صورتی امکان پذیر است که وابستگی بین دستورالعملهای اسمبلی نقض نشود. وابستگی ها در قالب وابستگی های داده ای و کنترلی امکان پذیر می باشند. وابستگی های داده ای بین جمله S2 که بعد از S1 قرار دارد به سه دسته ذیل تقسیم می شوند:

۱- وابستگی واقعی: در S1 تعریف و در S2 استفاده از متغیر تعریف شده قرار دارد.
۲- ضد وابستگی: در S1 استفاده و در S2 تعریف متغیر استفاده شده قرار دارد.
۳- وابستگی خروجی: در S1 تعریف و در S2 مجدداً تعریف متغیر تعریف شده قرار دارد.

مشاهده می شود که وابستگیهای داده ای بواسطه دسترسی به مکان مشترک ایجاد می شوند. می توان با ایجاد گراف وابستگی و مرتب سازی توپولوژیک آن، دستورالعمل ها را آنچنان مرتب سازی نمود که با کمترین وقفه به صورت خط لوله ای به اجرا در آیند. گراف وابستگی را ابتدا در سطح بلاکهای اولیه ایجاد می کنند.

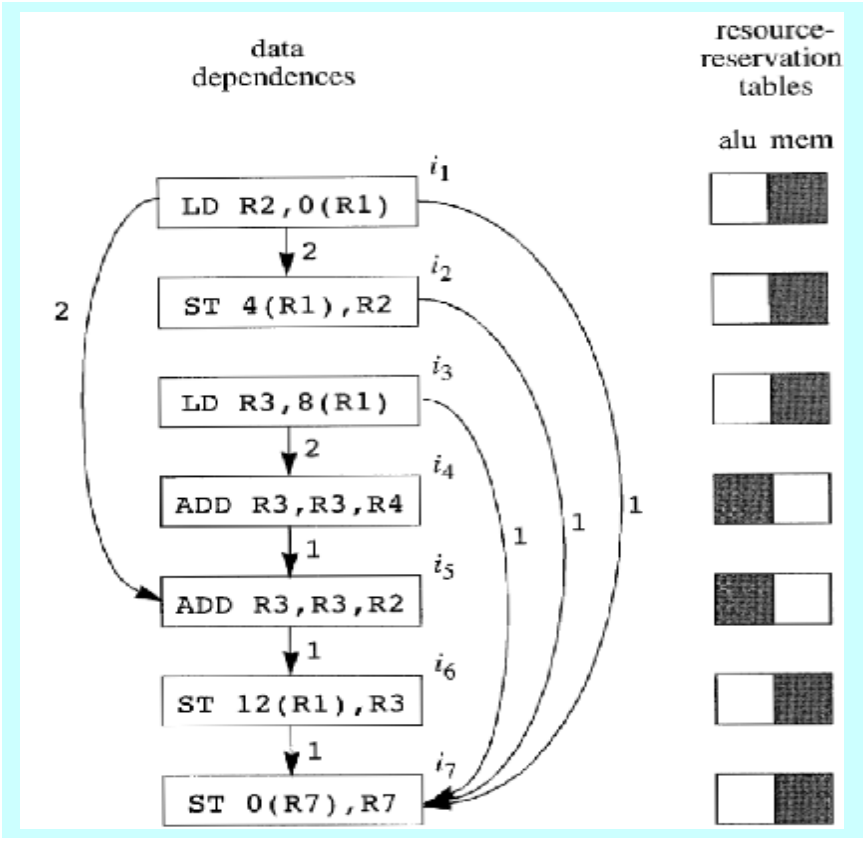
در گراف وابستگی هر گره شاخص یک دستورالعمل و منابع مورد نیاز مثل حافظه و واحد عملیاتی برای اجرای آن و لبه ها شاخص وابستگی بین دستورالعملها می باشند. هر لبه با تاخیر زمانی d_e که شاخص حداقل مدت زمانی است که گره مقصد پس از مبدا می تواند به اجرا در آید. برای نمونه ماشین ساده ای را در نظر بگیرید که در هر کلاک می تواند دو دستورالعمل را به اجرا درآورد. اولین دستورالعمل می بایست یک دستورالعمل پرش یا محاسباتی به صورت ذیل باشد:

OP dst, src1, src2

دومین دستورالعمل می تواند دستورالعمل بارکردن یا ذخیره سازی به صورت ذیل باشد:

LD dst, addr
St addr, src

دستورالعمل بارکردن LD دو سایکل طول می کشد. سایر دستورالعملها در یک سیکل زمانی اجرایشان کامل می شود. برای نمونه به شکل زیر توجه نمایید:



شکل ۵- ۳ گراف وابستگی زمانی برای قطعه کد ارایه شده در شکل قبلی

در شکل فوق دو منبع alu و mem برای هردستورالعمل و مقدار آدرس نسبی به صورت عدد صحیح قبل از نام ثبات حاوی آدرس مبنا مشخص شده است.

۱. شرح مساله

هدف در این تحقیق بررسی روشهای توزیع اتوماتیک کد برنامه ها می باشد. در این راستا بخصوص تشخیص توازی در انجام تکرارهای حلقه ها بسیار مطرح می باشد. نیاز به کامپیوترها با توان محاسباتی زیاد در جهت حل مسایل علمی و از سوی دیگر پیچیدگی برنامه نویسی برای سیستمهای موازی انگیزه ایجاد سوپرکامپایلرها جهت تشخیص اتوماتیک توازی در اجرای کد برنامه های ترتیبی و توزیع کد در سطح سیستمهای چند پردازنده ای را مطرح می نماید. عوامل پیچیدگی برنامه ها برای سیستمهای موازی را می توان به صورت ذیل خلاصه نمود:

۱- تشخیص جملاتی که به صورت موازی قابل اجرا هستند

۲- تشخیص تکرارهای حلقه ها با قابلیت اجرای موازی

۳- دانه بندی مناسب برای وظایف توزیع شونده بر روی پردازنده ها

۴- زمانبندی وظایف

۵- ایجاد کد قابل اجرا شونده بر روی چند پردازنده به صورت همزمان

لازمه ایجاد کد موازی برای اجرای هر چه سریعتر برنامه ها، حل مساله توزیع داهیانه کد برنامه در بین پردازنده ها است به قسمی که در تمام مدت اجرای برنامه کلیه پردازنده ها بطور همزمان فعال باشند و در عین حال هیچیک از پردازنده ها در انتظار گرفتن نتایج از پردازنده های دیگر قرار نگیرد و به عبارت دیگر وظایف توزیع شده حداقل وابستگی را با یکدیگر داشته باشند.

توازی در کد برنامه ها اصولا" بر مبنای تحلیل وابستگیها در دو سطح انجام می گیرد. در یک سطح وابستگی در بین دستورالعملهای داخل متن برنامه ترتیبی مورد نظر تشخیص داده می شود. در سطح دیگر وابستگی در بین تکرارهای حلقه تشخیص داده می شود.

مساله تشخیص وابستگی بین جملات منجر به تشخیص دو نوع وابستگی داده ای و کنترلی در بین جملات است. هنگتامیکه در یک جمله مقداری محاسبه شود که در جمله دیگر مورد استفاده قرار گیرد مسلما" این دو جمله به صورت موازی قابل اجرا نیستند و در اصطلاح وابستگی داده ی در بین این دو جمله وجود دارد. مشکل در اینجا تشخیص اسامی مستعار است زیرا، ممکن است یک متغیر از طریق اسامی دیگر مورد دسترسی قرار گیرد. وابستگی کنترلی زمانی در بین جمله ها مشاهده می شود که در یک جمله برای اجرای جمله دیگر تصمیم گیری شود. حاصل تشخیص وابستگیها در قالب گراف وظایف مشخص می شود. در گراف وظایف هر گره نمایانگر یک دستورالعمل و لبه ها شاخص دو نوع وابستگی داده ای و کنترلی در بین دستورالعملها می باشند. در اینجا مساله اصلی دانه بندی و زمانبندی وظایف است به قسمی که بتوان در ضمن پیمایش گراف، وظایف مستقل را به صورت همزمان با رعایت توازن بارکاری و حداقل هزینه ارتباطی بر روی پردازنده های متفاوت به صورت موازی به اجرا در آورد.

موازی سازی حلقه ها خود در چهار مرحله امکانپذیر است. در اولین مرحله، تحلیل وابستگی داده ای بین تکرارهای حلقه انجام می شود. در واقع یک تکرار در صورتی وابسته به تکرار دیگر است که مقداری را که در تکرار قبلی محاسبه شده مورد استفاده قرار دهد. در اینجا یک مساله تشخیص مقادیری که با تکرار حلقه تغییر پیدا می کنند، می باشد. به این نوع متغیرها در اصطلاح لفظ متغیرهای استنتاجی[۲] اطلاق می شود. متغیر استنتاجی در واقع مقدارش از تکرارهای حلقه منتج می شود. سوال اینجا است که چگونه می توان بر اساس شماره تکرار حلقه مقدار این دسته از متغیرها را محاسبه نمود.

وابستگی در بین تکرارهای حلقه بخصوص در اندیس آرایه ها مشهود است. اگر خانه ای از یک آرایه که مقداری در آن در یک تکرار حلقه محاسبه می شود در یکی از تکرارهای بعدی مورد استفاده قرار گیرد بین این دو تکرار مسلما" وابستگی داده ای وجود دارد. دسترسی به خانه های آرایه معمولا" از طریق اندیس آرایه انجام می شود. می بایست عباراتی که اندیس خانه آرایه را در دو تکرار مورد نظر محاسبه می کند مساوی با یکدیگر قرار داد. به این ترتیب از مساوی قرار دادن این نوع عبارات دستگاه معادلات خطی و یا غیر خطی وابسته به نوع عبارات ایجاد می شود [۱۱، ١٤، ٢٧، ٥٦، ٦٤، ٨٥ و ٩٣]. از حل این دستگاه معادلات تکرارهای وابسته آرایه مشخص می شوند. در اینجا مساله حل دستگاه معادلات است. بخصوص حل دستگاه معادلات غیر خطی می تواند مساله ای بسیار پیچیده باشد.

بعد از اینکه تکرارهای وابسته مشخص شدند مساله دیگر ایجاد حلقه موازی می باشد. ممکن است تعداد تکرارهای مستقل هزاران عدد باشد. در این صورت مساله تعیین شماره تکرارهای مستقل است. آیا باید این شماره ها را از یک فایل خواند که این مغایر با افزایش سرعت اجرایی کد است. بهترین راه حل بدست آوردن رابطه ای است که بتوان با استفاده از آن شماره تکرارهای مستقل را بدست آورد. با استفاده از

[2] Induction

روش هایی مثل حل معادله های دیوفانتین[3] [۱۱، ۸۵ و ۹۳]، تکرارهای وابسته حلقه بر اساس مجموعه ای از بردارهای پایه توصیف می شود [۵۶]. با استفاده از این بردارها می توان وابستگی بین تکرارهای حلقه را مشخص نمود. البته با استفاده از این روش برخی از تکرارهای غیر وابسته نیز به صورت وابسته تجلی می کنند که این در واقع مشکل این روش است.

پس از تشخیص وابستگیها در بین تکرارهای حلقه و تعیین بردارهای پایه که ازترکیب آنها بردارهای وابستگی ایجاد می شود مساله ایجاد حلقه های موازی و سریال با در نظر گرفتن وابستگیهای بین تکرارها است. با در دست داشتن فضای وابستگی می بایست تکرارهای وابسته را دسته بندی نمود و هر دسته از تکرارهای وابسته را در قالب یک حلقه ترتیبی یا در اصطلاح DoAcross و دسته های غیر وابسته را در قالب حلقه های موازی یا در اصطلاح DoAll مشخص نمود. مساله تعیین اندازه و تکرارهای وابسته موجود در داخل این دسته می باشد. مسلما" تکرارهای داخل هر دسته که قرار است به صورت ترتیبی به اجرا در آیند وابسته به یکدیگر می باشند و ترجیحا" فاصله اندیس این تکرارها باید قابل محاسبه در داخل یک حلقه باشد. با در نظر گرفتن اینکه وابستگیها به صورت منظم بر اساس بردارهای پایه وابستگی در بین تکرارهای حلقه در فضای وابستگی یکنواخت مشخص شده اند، می بایست فضای هندسی و چند بعدی وابستگیها را با در نظر گرفتن اندازه حافظه پردازنده‌ها طوری دسته بندی نمود که وابستگی تکرارهای درون دسته‌ها حداکثر ممکن و در بین دسته ها حداقل ممکن باشد.

یک روش بسیار محدود تقسیم بندی فضای تکرارها به اجزایی کاملا" یکسان به نام کاشی است. در این روش فضای یکنواخت وابستگی تکرارهای حلقه به دسته هایی به نام کاشی[4] تقسیم می شود به قسمی که

[3] Diophantine equations
[4] Tile

کمترین وابستگی بین کاشی‌ها وجود داشته باشد [۲، ۳۲، ٤٤، ٦٦ و ۹۳]. هدف از *کاشی بندی*[5] یافتن شکل و اندازه کاشی مناسب جهت موازی سازی حلقه هاست [٤٧، ۷۱ و ۷۲]. یک *کاشی*، در برگیرنده مجموعه ای از تکرارهای حلقه است که باید بر روی یک پردازنده اجرا شوند. مسئله اصلی تعیین اندازه و شکل بهینه کاشی است؛ به قسمی که (۱) بتوان هر کاشی را با توجه به منابع موجود یک پردازنده، روی آن به طور پیوسته[6] و ترتیبی اجرا کرد. (۲) میزان ارتباط بین پردازنده ها کمینه شود. (۳) هزینه های سرباری شامل ارتباط های ورودی/خروجی، میزان حافظه مصرفی محلی و غیره به کمترین مقدار ممکن کاهش داده شود. این یک مسئله بهینه سازی از نوع NP–سخت است [۵، ۱۵،۳۲، ۳٦، ۳۸، ٤۰، ٤٤، ٥٦، ٦٤، ٦٦، ۷۸، ۸۵ و ۹۳].

پس از کاشی بندی فضای یکنواخت وابستگی بین تکرارهای حلقه ها مساله اصلی تعیین کاشی های مستقل است که می توانند به صورت موازی به اجرا در آیند. در این راستا الگوریتم های موجود در این زمینه بیشتر بر دو رویکرد ایجاد موج ها[7] و یا دنباله ای از تکرارهای وابسته[8] استوار هستند [۳، ۲۵، ۳۲، ۵۲ و ٦٦]. در رویکردی که به وسیله گوماس[9] و آتاناساکی[10] [۳۱] ارایه شده است، از روش ایجاد موج ها استفاده می شود.

نهایتا"، کد موازی حاصل از حلقه های ترتیبی تو در تو می بایست بر روی یک معماری موازی زمان بندی[11] می شود. برای این منظور نیز روش های مختلفی برای کاهش زمان اجرای تکرارهای حلقه به طور موازی ارایه شده است [۷، ۲۱، ۲۵، ۳۲، ٤۱، ۵۲، ٦۱، ٦٦، ۷۰ و ۷۷]. این روشها مبتنی بر حذف

[5] Tiling
[6] Atomic
[7] Wave-fronts
[8] Skew
[9] Goumas
[10] Athanasaki
[11] Scheduling

ارتباط های بین پردازنده ای و با تخصیص کاشی های وابسته به پردازنده های یکسان می باشد [۵۲]. یکی از این رویکردها به وسیله آتاناساکی [۸] ارایه گردیده است. حال اگر کد ترتیبی داخل بدنه حلقه ها در نظر گرفته شود، آنگاه کد موازی حاصل از فضای تکرار کاشی شده به همراه کد داخل بدنه حلقه می توان به عنوان یک گراف وظایف در نظر گرفته شود. آنگاه یکی از مسایل NP–سخت، انجام زمان بندی برنامه موازی حاصل از کد ترتیبی بر روی یک سیستم چند پردازنده ای است [۲۸]. در این سیستم ها زمان بندی موثر برای اجرای یک برنامه موازی جهت نایل شدن به کارآیی بالا، امری حیاتی است. این زمان بندی باید به گونه ای انجام گیرد که بتوان زمان اجرای کل برنامه را با توجه به زمان وظایف و ارتباط بین پردازنده ها، کمینه نمود. از آنجایی که مسئله زمان بندی گراف وظایف یک مسئله NP–سخت است [۲۸]؛ رویکردهای مبتنی بر روش های قطعی در این زمینه کارآیی چندانی نخواهند داشت. استفاده از رویکردهای تکاملی و به طور عمده الگوریتم های ژنتیک[۱۲] با توجه به ماهیت غیر قطعی که دارند؛ برای حل این مسئله موثر می باشد.

۲. تئوری وابستگی

وابستگیها از اجرای موازی ممانعت می کنند. وابستگیها در بین جملات و در بین تکرارهای حلقه ها در متن برنامه ها مطرح هستند. استقلال جملات و تکرارهای حلقه از یکدیگر امکان اجرای موازی و همزمان آنها را با یکدیگر فراهم می نماید. لذا، جهت تبدیل اتوماتیک کد سریال به موازی در اولین گام می بایست

[12] Genetic algorithms (GAs)

وابستگیها را تشخیص داد. در ادامه در بخش ۱-۲ وابستگی بین جملات و در بخش ۲-۲ وابستگی بین تکرارهای حلقه مطرح شده است.

۱-۲ وابستگی بین جملات

وابستگیها در بین جملات در واقع قیوداتی هستند که موجب می شوند دستورالعملهای برنامه با ترتیب خاص بر طبق منطق برنامه به اجرا در آیند. وجود این نوع قیودات مانعی در اجرای موازی جمله ها در داخل برنامه‌ها است. اصولاً دو نوع وابستگی در کد برنامه ها و در بین جملات برنامه وجود دارد. یک دسته وابستگیهای داده ای می باشند وابستگی داده ای بواسطه استفاده جمله وابسته از مقدار محاسبه شده در جمله ای دیگر بوجود می آید. برای نمونه به قطعه کد ذیل توجه نمایید:

S_1 PI=3.14

S_2 R=5.0

S_3 AREA=PI*R**2

از آنجائی که S_3 از داده ای استفاده می نماید که S_1 و S_2 تولید کرده اند، بنابراین از S_1 و S_2 به S_3 یک وابستگی ایجاد می شود. این وابستگی دو جمله S_1 و S_2 را مقید به اجرا قبل از S_3 می نماید.

وابستگی کنترلی نوع دیگری از وابستگی است که برای حفظ ترتیب اجرای دستوراتی که منطقا به یکدیگر وابسته می باشند مطرح است. برای نمونه در قطعه کد ذیل S_2 نمی تواند قبل S_1 اجرا شود زیرا اجرای S_2 وابسته به اجرای S_1 می باشد.

S_1 IF (T.NE.10) GOTO S_{10}

S_2 A=A/T

S_3 Continue

تعریف ۱: وابستگی داده ای در صورتی بین جمله های S_1 و S_2 وجود دارد که اولا"، هر دو جمله به یک مکان حافظه دسترسی پیدا کنند و یکی از آنها ذخیره سازی باشد. ثانیا"، یکی قبل از دیگری اجرا شود. ثالثا"، یک مسیر اجرایی ممکن از S_1 به S_2 وجود داشته باشد [۵۶ و ۹۳].

برای تشخیص وابستگیهای داده ای زنجیره های تعریف و استفاده ایجاد می شود. این نوع زنجیره ها در واقع محل مقدار دهی به یک متغیر با محلهای استفاده از آن مقدار را مرتبط می کنند. مشکل اسامی مستعار وتشخیص پارامترهای مرجع در فراخوانیها است. برای تشخیص وابستگیهای کنترلی نقاط تصمیم گیری در داخل گراف جریان کنترلی می بایستی مشخص شوند. ترکیب را گراف وظایف می نامند. تشخیص وابستگیها جز مباحث کلاسیک در کتب کامپایلر است. دو نکته اصلی دانه بندی و زمانبندی گراف وابستگیها به نام گراف وظایف است.

وابستگی داده ای به سه طریق واقعی[۱۳]، متضاد[۱۴] و خروجی[۱۵] ممکن است در کدبرنامه ها مشاهده شود: [۵۶ و ۹۳]. برای نمونه به قطعه کد ذیل توجه نمایید:

```
S1:   R := 55;
S2:   S := R * 415;
S3:   R := 12 / (D + E)
S4:   R := (S * 3.8) / 2.71
```

[13] – True Dependence

[14] Anti dependednce

[15] Output

در قطعه کد فوق، S_2 به S_1 *وابستگی واقعی* دارد. یعنی S_1 باید قبل از S_2 اجرا شود. هم چنین S_2 *وابستگی* عکس به S_3 دارد؛ یعنی اگر جای دو دستور عوض شود، وابستگی واقعی ایجاد خواهد شد و در نهایت بین S_3 و S_4 *وابستگی خروجی* وجود دارد؛ یعنی با جا به جایی این دو دستور خروجی برنامه فرق می کند. وابستگی های برعکس و خروجی را می توان با اعمال تبدیل های مناسب از جمله تغییر نام متغیر از بین برد. این گونه از وابستگی ها بیشتر در زبان های امری[16] مشاهده می گردند. به همین خاطر به این نوع از وابستگی ها، *وابستگی های مصنوعی*[17] گویند.

الف- وابستگی جریان[18] یا وابستگی واقعی

وقتی که در یک تکرار حلقه مقداری به متغیری نسبت داده شود و در یکی از تکرارهای بعدی مورد استفاده یا خواندن قرار می گیرد وابستگی واقعی بین این دو تکرار وجود خواهد داشت. در قطعه کد ذیل وابستگی از دستور S1 به خودش وجود دارد زیرا مقدار نسبت داده شده به متغیر A(I+1) در تکرار بعدی حلقه مورد استفاده قرار می گیرد و می نویسیم S1 δ S1.

```
      do I = 1, N-1
S1:     A(I+1) = A(I) + B(I)
      enddo
```

چنانچه جمله S_1 مقداری را ذخیره کند و جمله S_2 آن مقدار استفاده نماید در اینصورت وابستگی *واقعی* بین این دو جمله وجود دارد.

[16] Functional languages
[17] Artificial dependence
[18] Flow- Dependence

$$S_1 \qquad X = \ldots$$
$$S_2 \qquad \ldots = X$$

این نوع وابستگی تضمین می نماید که جمله دوم مقداری را دریافت می کند که توسط جمله اول تولید شده است. این نوع وابستگی را وابستگی جریانی نیز می نامند و با علامت S_1 & S_2 نمایش داده می شود. قرار داد نمایش گرافیکی این وابستگی فلش جهت داری از جمله منبع که مقدار مورد استفاده را ایجاد می کند به سوی جمله ای که مقدار را استفاده می کند، است.

ب- ضد وابستگی[19]

وقتی که درتکرار فعلی مقدار متغیری خوانده شود که در یکی از تکرارهای بعدی در آن متغیر مقداری نوشته می شود و ابستگی را ضد وابستگی یا وابستگی متضاد می نامند. در قطعه کد ذیل یک ضد وابستگی بین S1 با S2 وجود دارد زیرا متغیر B(I,J) در S1 استفاده می شود ولی در دستور بعدی مجدداً مقدار دهی می گردد (در همان تکرار حلقه) که می نویسیم: S1 δ' S2 .

```
      do I = 1, N
        do J = 1, M
S1:       A(I,J) = B(I,J) + 1
S2:       B(I,J) = C(I,J) -1
        enddo
      enddo
```

در صورتیکه جمله اول از مکانی مقداری را بخواند که جمله دومی در آن مقدار ذخیره نماید، ضد وابستگی یا وابستگی متضاد بین دو جمله وجود دارد.

$$S_1 \qquad .. = X$$
$$S_2 \qquad X = \ldots$$

[19] Anti- Dependence

تعیین این نوع وابستگی در این جا سبب می‌شود که جای جمله S_2 و S_1 تعویض نگردد. اگر چنین اتفاقی بیفتد، S_1 از مقداری استفاده می نماید که توسط S_2 تولید شده است. در حقیقت این وابستگی بدین دلیل تعیین و ترسیم می گردد که یک وابستگی جدید از نوع ,واقعی که در برنامه اصلی موجود نمی‌باشد به وجود نیاید. این وابستگی با $S_1\delta^{-1}S_2$ نمایش داده می‌شود.

ج- وابستگی خروجی[20]

چنانچه هر دو جمله S_2 و S_1 در یک مکان مقدار بنویسند وابستگی از نوع خروجی در بین این دو جمله وجود دارد.

$S_1 \quad X=...$
$S_2 \quad X=...$

ترسیم این وایستگی سبب می شود که جای جمله S_1 و S_2 تعویض نگردد. در صورت تعویض این دو جمله ممکن است جملات بعدی مقادیر اشتباه را مصرف نمایند. برای مثال قطعه کد زیر را در نظر بگیرید :

S_1 X=1
S_2 …..
S_3 X=2
S_4 W=X*Y

در اینجا اگر جای S_1, S_2 عوض شودمقدار W به اشتباه تغییر می‌کند. در سیستمهای خط لوله[21] این وابستگی خروجی را مخاطره یا در اصطلاح Hazard می نامند. وابستگی واقعی را

[20] Output- Dependence
[21] Pipeline

RAW Hazard (read after write) و وابستگی متضاد را WAR Hazard یا (Write after read) می‌باشد. وابستگی خروجی را WAW Hazard نیز می نامند.

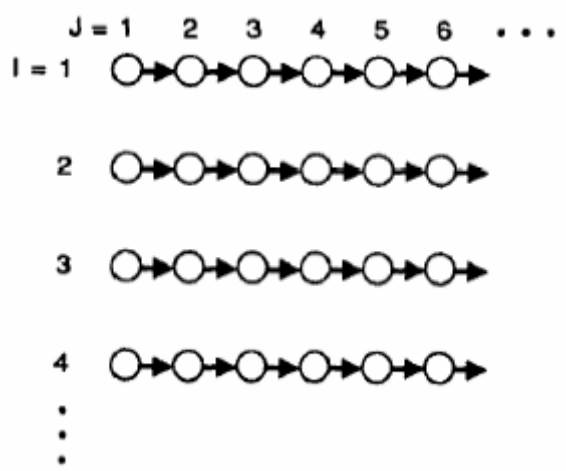
وقتی که در یک تکرار حلقه متغیری در دو دستور متوالیا" مقدار دهی شود نوع وابستگی بین تکرارهای حلقه وابستگی خروجی نامیده می شود. برای نمونه در قطعه کد زیر، وابستگی خروجی بین S2 و S1 وجود دارد زیرا متغیر B(I+1) در S2 مقدار دهی شده که ممکن است در تکرار بعدی حلقه توسط S1 مقدار دهی مجدد شود. این نوع وابستگی به صورت S2 δ^o S1 مشخص می شود.

```
     do I = 1, N-1
S1:    if(A(I) > 0) B(I) = C(I)/A(I)
S2:    B(I+1) = C(I) / 2
     enddo
```

اغلب، وابستگی ها دارای یک فاصله ثابت در هر بعد از فضای تکرار هستند که آنرا با بردار فاصله[22] نمایش می دهند. مثلا برای قطعه کد ذیل، وابستگی ها مطابق با شکل ۱ نمایش داده می شوند. هر تکرار (i,j) به مقدار محاسبه شده در تکرار (i,j-1) وابسته است. فاصله وابستگی در حلقه i برابرصفر و در حلقه j یک می باشد.

```
     do I = 1, N
       do J = 2, M
S1:      A(I,J) = A(I,J-1) + B(I,J)
       enddo
     enddo
```

[22] Distance Vector



شکل ۱- وابستگی تکرارهای حلقه

در اکثر تبدیلات حلقه ها علامت فاصله ها اهمیت بیشتری نسبت به مقدار فاصله دارد. این علامت را در بردار جهت[۲۳] نمایش می دهند. مثلا بصورت {-,0,+} که در گذشته بصورت {<,=,>} نمایش داده می شد. برای نمونه به قطعه کد ذیل توجه نمایید:

```
      do I = 1, N
        do J = 1, N
S1:       X(I+1,2*J) = X(I,J) + B(I)
        enddo
      enddo
```

در نمایش دیگری، فقط سطح تودرتو بودن خارجی ترین حلقه با فاصله مثبت را نگهداری می کنند. بعنوان مثال در برنامه ارایه شده در شکل ۳ فاصله تا حلقه خارجی تر صفر است ولی در حلقه داخلی تر فاصله مثبت دو می باشد پس می توان نوشت: $S2\ \delta^2\ S1$. این وابستگی را محل حمل شدن[۲۴] با حلقه داخلی j می دانیم. ممکن است که برخی از وابستگی ها حمل نشوند. مانند قطعه کد ذیل که ارجاع به A(I,J) وابستگی از S1 به S2 با فاصله صفر در هر حلقه دارد.

[23] Direction Vector
[24] Carried

```
      do I = 1, N
        do J = 2, M
S1:       A(I,J) = B(I,J) + C(I,J)
S2:       D(I,J) = A(I,J) + 1
        enddo
      enddo
```

در اینصورت گفته می شود که S2 $\delta^{(0,0)}$ S1 یا S2 $\delta^{(=,=)}$ S1 زیرا با هیچ یک از حلقه ها حمل نمی شود لذا گفته می شود استقلال حلقه ای وجود دارد و یا به عبارت دیگر S2 δ^{∞} S2 .

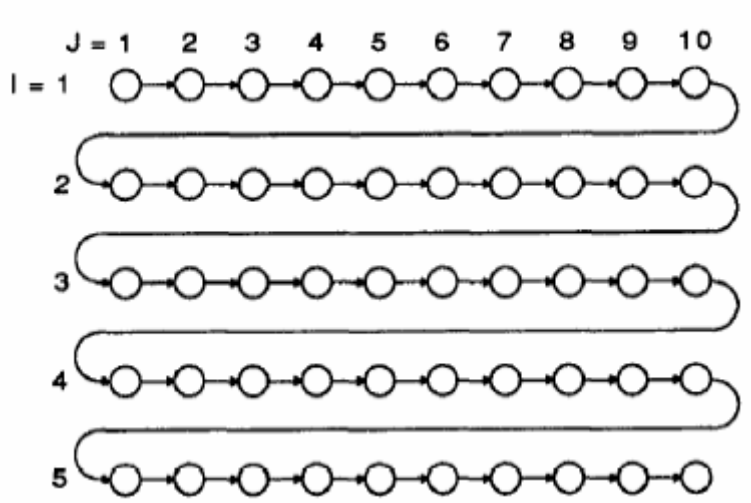
۲-۲ فضای تکرارهای حلقه

قبل از اینکه به بیان انواع وابستگیها در بین تکرارهای حلقه پرداخته شود مفهوم فضای تکرارهای حلقه را باید مشخص نمود. حلقه های تکرار تو در تو یک فضای تکرار[۲۵] که در واقع فضای کارتزین گسسته متناهی با ابعاد برابر با تعداد سطوح تو درتو بودن حلقه هاست ایجاد می کنند. مثلا دو حلقه برنامه ۱ فضای تکرار دو بعدی شکل ۲ را می سازند. منطق یک حلقه سریال بیان می کند که فضای تکرار چگونه پیمایش می شود. مثلاً، در شکل ۲ مسیر اجرایی از از چپ به راست و از بالا به پایین است.

```
do I = 1, 5
  do J = 1, 10
    A(I,J) = B(I,J) + C(I)*D(J)
  enddo
enddo
```

یک نمونه حلقه

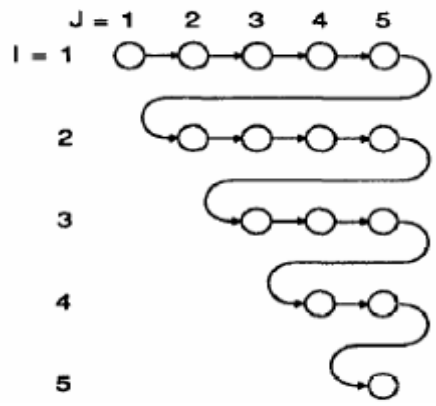
[25] Iteration Space



شکل ۲- فضای برداری تکرارهای حلقه

البته الزامی نیست که فضای تکرار مربعی باشد. در بسیاری از الگوریتمهای متداول حدود حلقه های داخلی به مقدار اندیس حلقه های خارجی تر وابسته است. مثلا فضای تکرار شکل ۳ که مربوط به کد ذیل می باشد مثلثی[۲۶] است. فضای تکرار می تواند اشکال دیگری مانند ذوزنقه[۲۷] و غیره نیز داشته باشد.

```
do I = 1, 5
  do J = I, 5
    A(I,J) = B(I,J) + C(I)*D(J)
  enddo
enddo
```



شکل۳- فضای تکرار مثلثی

[26] Triangular Loops
[27] Trapezoid

۳-۲ وابستگی بین تکرارهای حلقه

بسط مفهوم وابستگی به حلقه‌ها نیاز به این دارد که اندیس حلقه‌ها نیز در بیان وابستگی جملات به کار گرفته شود. برای نمونه در قطعه کد ذیل با در نظر گرفتن مقدار اندیس I و تاثیر آن در دسترسی به خانه های دو آرایه A و B مشاهده می شود که جمله S1 به خودش وابسته است.

$$\begin{aligned} &Do\ I = 1, N \\ &S_1 \qquad A(I+1) = A(I) + B(I) \\ &ENDDO \end{aligned}$$

به عبارت دیگر برای نمونه مقدار محاسبه شده در تکرار ۵ از حلقه برای A[6] وابسته به مقدار محاسبه شده در تکرار ۴ است. بنابراین هر تکرار این حلقه وابسته به تکرار قبلی آن می باشد. در مثال ذیل هر تکرار حلقه به مقداری که در دو تکرار قبل تولید می‌شود وابستگی دارد در صورتی که در قطعه کد اولی هر تکرار به مقداری که در تکرار قبلی آن تولید می‌شود وابسته است.

$$\begin{aligned} &DO \qquad I = 1, N \\ &S1 \qquad A(I+2) = A(I) + B(I) \\ &ENDDO \end{aligned}$$

بیان دقیق وابستگی در حلقه ها نیاز به این دارد که اندیس حلقه ها نیز به نوعی پارامتریک شود تا وابستگی بین جملات براساس اندیس حلقه ها بیان گردد. بدین منظور برداری از اعداد صحیح که نمایش از شماره تکرار حلقه ها در یک حلقه آشیانه ای می باشد ساخته می شود. برای مثال به حلقه ذیل توجه نمایید:

$$\begin{aligned} &DO \qquad I = 1, N \\ &\quad ..... \\ &ENDDO \end{aligned}$$

در این مثال شماره هر تکرار برابر با اندیس حلقه می باشد. برای تکرار اول، شماره تکرار یک. برای تکرار دوم، شماره تکرار ۲ و غیره می باشد. اما در قطعه کد ذیل به این ترتیب نمی توان عمل نمود:

$$DO \quad I = 1, u, s$$
$$.....$$
$$ENDDO$$

در تکرار دوم از مثال فوق مقدار شمارنده حلقه برابر با S + 1 است. در بسیاری از موارد استفاده از شماره تکرار نرمال شده مورد استفاده قرار می گیرد. این شماره نرمال شده از ۱ شروع می شود و نهایتا" مقدارآن مساوی با تعداد تکرارهای حلقه می شود. تا یک حد بالا اجرا می شود.

تعریف ۲- برای یک حلقه دلخواه که با اندیس I از L تا U با گام هایی به طول S اجرا می شود، شماره تکرار نرمال شده یک تکرار خاص I برابر با مقدار $(I - L + S) / S$ می باشد.

در یک حلقه تودرتو، عمق آشیانه ای حلقه برابر با تعداد حلقه های در بر گیرنده آن حلقه بعلاوه یک می باشد. به عبارت دیگر، حلقه ها از خارجی ترین حلقه تا داخلی ترین اندیس شده اند که این اندیس از یک شروع می شود.

تعریف ۳- با فرض n حلقه تو در تو، بردار تکرار I برداری از اعداد صحیح می باشد که شامل شماره تکرار برای هر کدام از حلقه ها به ترتیب سطح تو در تویی می باشد. به عبارت دیگر بردار تکرار با $i = \{i_1, i_2, ..., i_n\}$ نمایش داده می شود به قسمی i_R برای $1 \le R \le n$ نمایش شماره تکرار برای حلقه در سطح R می باشد. برای نمونه در قطعه کد زیر S[2,1] شاخص جمله S در تکرار ۲ از I و تکرار ۱ برای J را نشان می دهد.

$$Do\ I = 1,2$$
$$Do\ j = 1,2$$
$$S$$
$$ENDDo$$
$$ENDDo$$

مجموعه همه بردارهای تکرار برای یک حلقه، فضای تکرار یا Iteration Space را برای آن حلقه مشخص می نماید. برای مثال برای جمله S در قطعه کد فوق، فضای تکرار $\{(1,1)(1,2)(2,1)(2,2)\}$ می باشد.

به دلیل اهمیت ترتیب اجرای وابستگی،یک بردار تکرار باید براساس ترتیب اجرای حلقه های مربوطه مرتب شود. فرض نمایید که i یک بردار تکرار و i[1:k] بردار k تایی که شامل k عنصر سمت چپ i می باشد، است. ترتیب لغوی یا در اصطلاح Lexicographic بردارهای تکرار به طول n به صورت ذیل تعریف می شود.

تعریف ۴- تکرار i بعد از تکرار j می آید اگر و تنهااگر

۱- $i[1:n-1] < j[1:n-1]$ و یا

۲- $i[1:n-1] = j[1:n-1]$ و $i_n < j_n$

قضیه ۱- وابستگی حلقه

یک وابستگی از جمله S_1 و به جمله S_2 در یک حلقه وجود دارد، اگر و تنها اگر دو بردار تکرار j و i وجود داشته باشد به قسمی که:

(۱) i<j یا i=j و یک مسیر از S_1 به S_2 در بدنه حلقه وجودد داشته باشد.

(۲) عبارت S_1 به مکان M حافظه در تکرار i دسترسی پیدا می کند و S_2 در تکرار j به مکان M دسترسی پیدا می کند.

(۳) یکی از دسترسی ها نوشتن باشد.

تعریف ۵ – دو محاسبه هم ارز می باشند اگر با ورودی های یکسان، مقدارهای یکسان برای متغیرهای خروجی تولید کنند و عبارت خروجی با یک ترتیب خاص تولید شود.

تعریف ۶- تبدیل ترتیب مجدد یا در اصطلاح reordering به هر نوع جابجایی و تبدیل در کد برنامه برنامه اتلاق می شود که منحصراً ترتیب اجرای کد را تغییر دهد بدون اینکه اجرای عبارتی را اضافه و یا کم کند.

تعریف ۷ – هر تبدیل ترتیب مجدد یک وابستگی را حفظ می کند اگر ترتیب اجرای منبع و مقصد وابستگی را تغییر ندهد.

قضیه ۲- قضیه اساسی وابستگی

هر تبدیل ترتیب مجدد که همه وابستگی های یک برنامه را حفظ می کند مفهوم برنامه را حفظ می کند.

تعریف ۸- یک تبدیل برای یک برنامه معتبر است چنانچه در صورت اعمال آن بر کد برنامه، همه وابستگی ها را حفظ کند.

تعریف ۹- فرض کنید یک وابستگی از عبارت S_1 در تکرار i از حلقه تو در تو به عبارت S_2 در تکرار j ام وجود دارد، در این حالت بردار مسافت وابستگی d(i,j) به صورت برداری با طول n شاخص تفاضل تکرار i از j و به صورت ذیل تعریف می شود:

$$d(i.j)_k = j_k - i_R \qquad , \forall k \in [1..n]$$

تعریف ۱۰- فرض کنید یک وابستگی از عبارت S_1 در تکرار i یک حلقه به عبارت S_2 در تکرار j وجود دارد، بنابراین بردار جهت وابستگی d(i,j) به عنوان برداری با طول n که در آن n درجه تو در تویی حلقه می باشد، تعریف می شود به گونه ای که:

$$d(i.j)_k = \begin{cases} "<" & if \quad d(i.j)_k > \phi \\ "=" & if \quad d(i.j)_k = \phi \\ ">" & if \quad d(i.j)_k < \phi \end{cases}$$

مثال:

```
DO    I = 1, N
      DO    I = 1, M
            DO    I = K = 1 + L
S1                  A(I + 1, J, K - 1) = A(I, J, K) + 10
            ENDDO
      END
ENDDO
```

عبارت $S1$ یک وابستگی ,واقعی به خودش دارد که بردار جهت آن $(<,=,>)$ می باشد بدین مفهوم که اندیس حلقه خارجی در منبع کوچکتر از اندیس مقصد می باشد. اندیس حلقه میانی در منبع و مقصد یکسان است و در حلقه داخلی اندیس منبع بزرگتر از اندیس مقصد وابستگی می باشد.

در این حلقه تو در تو در مجموع ۹۰ وابستگی مختلف وجود دارد. در عمل، هیچ کامپایلری نمی تواند این تعداد وابستگی را برای هر جمله در یک حلقه نگهدارد. بنابراین باید روشی را برای خلاصه کردن وابستگی ها یافت. بدین منظور برای وابستگی های مختلف فقط

بردارهای وابستگی متفاوت و متمایز نگهداری می شود. اگرچه با این کاهش باز هم ممکن است تعداد زیادی بردار وابستگی وجود داشته باشد. برای مثال به قطعه کد ذیل توجه نمایید.

```
DO      j = 1,10
        DO      I = 1,99
S1                  A(I + J) = A(i, J) + X
S2                  C(I + J) = A(100 – i, J) + Y
                ENDDO
ENDDO
```

براساس تئوری وابستگی، برای این که S_2 وابسته داده ای به S_1 باشد باید دو شرط برقرار باشد. اولا"، یک مسیر اجرایی وجود داشته باشد که در آن هم S_1 و S_2 به یک مکان حافظه M دسترسی پیدا کنند. ثانیا"، اجرای S_1 که به مکان M دسترسی پیدا می کند قبل از S_2 باشد. برای این سناریو دو حالت متصور است:

۱- S_1 و S_2 در یک تکرار به M دسترسی پیدا کنند

۲- S_1 در یک تکرار به M دسترسی پیدا کند و S_2 در تکرارهای بعدی حلقه به این مکان دسترسی پیدا کند.

حالت اول را وابستگی مستقل از حلقه و حالت دوم را وابستگی وابسته به حلقه یا در اصطلاح Loop Carried dependence گویند. وابستگی وابسته به حلقه در بین تکرارهای حلقه به وجود می آید: برای مثال:

```
DO      I = 1, N
S1              A(I + 1) = F(I)
S2              C(I + 1) = A(I)
ENDDO
```

در ایـن مثـال در هـر تکـرار حلقـه I بـه جـز تکـرار اول، S_2 از مقـداری از آرایـه A اسـتفاده مـی نماید کـه توسـط S_1 تولیـد شـده اسـت. بـه عبـارت دیگـر نـوع وابسـتگی خوانـدن بعـد از نوشـتن یـا بطـور خلاصـه (RAW) اسـت. بنـابراین یـک وابسـتگی true از S_1 و S_2 وجـود دارد. S_2 مقـدار تولیـد شـده توسـط S_1 را در یـک تکـرار بعـدی حلقـه اسـتفاده مـی نمایـد پـس ایـن وابسـتگی، وابسـته بـه حلقـه مـی باشـد. همچنـین S_2 مقـداز F را در یـک تکـرار تولیـد کـرده و همـان مقـدار را S_1 در تکـرار بعـدی مصرف می نماید بنابراین یک وابستکی true از نوع وابسته به حلقه از S_2 به S_1 وجود دارد.

نکتـــه: اگـر هـر کـدام از تکرارهـای حلقـه بـه تنهـایی انتخـاب و اجـرا شـوند هـیچ وابسـتگی وجـود نخواهد داشت.

تعریـــف ۱۱- جملـه S_2 یـک وابسـتگی وابسـته بـه حلقـه بـه جملـه S_1 دارد اگـر و تنهـا اگـر S_1 بـه مکـان M در تکـرار i دسترسـی پیـدا کـرده و S_2 در تکـرار j بـه همـان مکـان دسترسـی پیـدا کنـد و $d(i, j) > \phi$ باشـئد و همچـین d(i,j) شـامل یـک "<" بـه عنـوان سـمت چـپ تـرین عنـصر کـه "=" نیـست باشد.

تعریـــف ۱۲- یـک وابسـتگی ناشـی از حلقـه از عبـارت S_1 بـه S_2، رو بـه عقـب یـا در اصـطلاح backward مـی باشـد اگـر S_2 قبـل از S_1 در بدنـه حلقـه ظـاهر شـود و یـا ایـن کـه هـر دو در یـک عبـارت باشد. یک وابستگی forward می باشد اگر S_2 بعد از S_1 در بدنه حلقه ظاهر شود.

تعریف ۱۳- سطح یک وابستگی ناشی از حلقه اندیس سمت چپ ترین غیر مساوی "=" از D(i,j) برای وابستگی می باشد. برای مثال قطعه کد ذیل را در نظر بگیرید.

```
DO    I = 1,10
      DO    J = 1,10
            DO    K = 1,10
                  A(I, J, K+1) = A(I, J, K)
            ENDDO
      ENDDO
ENDDO
```

D(i,j) در این مثال $(<,=,=)$ می باشد بنابراین سطح وابستگی ۳ می باشد.

تعریف ۱۴- یک وابستگی برآورده می شود، اگر تبدیلاتی که نمی توانند آن را نگهدارند منع شوند.

قضیه ۴- هر تبدیل ترتیب مجدد یا در اصطلاح reordering که (۱) ترتیب تکرار حلقه سطح k را نگه می دارد.(۲) هیچ حلقه ای در سطح پایین تر از k به درون سطح k نیاید.(۳) هیچ حلقه ای در سطح بزرگتر از k با موقعیتی خارج از سطح k تعویض نگردد همه وابستگی های سطح k را نگه می دارد.

برای نمونه به مثال های ذیل توجه فرمایید:

```
DO    I = 1,10
S1          A(I+1) = F(I)
S2          F(I+1) = A(I)
```

این حلقه معادل با حلقه ذیل است :

```
      DO    I = 1,10
S2          F(I+1) = A(I)
S1          A(I+1) = F(I)
```

علت معادل بودن دو حلقه فوق این است که وابستگی ها در سطح ۱ می باشند و ترتیب تکرارها در سطح ۱ حلقه تغییری نکرده است. البته تبدیل ذیل مجاز نمی باشد.

$$\begin{array}{lll} & DO & I=10,1 \\ S_1 & & A(I+1)=F(I) \\ S_2 & & F(I+1)=A(I) \end{array}$$

علت این است که ترتیب اجرای تکرارهای سطح یک که وابستگی ها از آن منتج می شوند را تغییر می دهد. به عنوان نمونه ای دیگر به قطعه کد ذیل توجه فرمایید.

$$\begin{array}{lll} DO & I=1,10 & \\ & DO & J=1,10 \\ & & DO \quad K=1,10 \\ S & & A(I+1,J+2,K+3)=A(I,J,K)+B \\ & & ENDDO \\ & ENDDO & \\ ENDDO & & \end{array}$$

از آنجائی که وابستگی ها در سطح ۱ می باشند بنابراین در طرح بالاتر می توان حلقه ها را با یکدیگر جا نمود و همچنین ترتیب اجرای هر حلقه را نیز تغییر داد.

$$\begin{array}{lll} DO & I=1,10 & \\ & DO & K=1,10,-1 \\ & & DO \quad J=1,10 \\ S & & A(I+1,J+2,K+3)=A(I,J,K)+B \\ & & ENDDO \\ & ENDDO & \\ ENDDO & & \end{array}$$

وابستگی $S_1 \ \delta_L \ S_2$ با S_2 و S_1 میان Level –L نمایش داده می شود. وابستگی مستقل از حلقه نتیجه موقعیت نسبی جملات می باشد. بنابراین وابستگی های مستقل از حلقه ترتیب

اجرای کد در درون حلقه های تو در تو را نشان می دهد ولی وابستگی ها وابسته به حلقه ترتیب اجرای حلقه ها را نمایش می دهد.

تعریف ۱۵- جمله S_2 یک وابستگی مستقل از حلقه به جمله S_1 دارد اگر و تنها اگر دو بردار وابستگی i و j وجود داشته باشد به نوعی که (۱) جمله S_1 به مکان حافظه M در تکرار i دسترسی پیدا کرده و جمله S_2 در تکرار j به این مکان دسترسی پیدا کند و i=j باشد و (۲) یک مسیر جریان کنترلی از S_1 به S_2 در درون تکرار وجود داشته باشد.

قضیه ۵- اگر یک وابستگی مستقل از حلقه از S_1 به S_2 وجود داشته باشد هر reordering ای که عبارات را میان تکرارها جایی نکنند و ترتیب اجرای S_1 و S_2 را در بدنه حلقه نگهدارد وابستگی ها را حفظ می کنند.

به طور کلی می توان گفت وابستگی ها مستقل و وابسته به حلقه مجموعه تمام وابستگی های داده را افراز می کنند. وجود وابستگی $S_1 \delta S_2$ بدین مفهوم است که S_1 قبل از S_2 اجرا شود که این موضوع در دو حالت روی می دهد،

۱- زمانی که بردار مسافت برای وابستگی بزرگتر از صفر باشد.

۲- زمانی که بردار مسافت برابر با صفر باشد و S_1 در درون متن حلقه قبل از S_2 رخ می دهد.

حالت اول معیار وابستگی وابسته به حلقه و حالت دوم وابستگی مستقل از حلقه را توصیف می نماید. یک تبدیل که تکرارهای سطح k را بدون هیچ تغییر دیگری reorder می کند معتبر است اگر حلقه هیچ وابستگی دیگری را حمل نکند.

قضیه ۶- فرض کنید α, β دو بردار تکرار در فضای تکراری زیر باشند.

$DO\ i_1 = L_1, U_1, S_1$

$DO\ i_2 = h_2, U_2, S_2$

...

$DO\ i_n = L_n, U_n, S_n$

$A_1 F_1(i_1,...,(n),...,F_m(i_1,..,i_n)) = ...$

$... = A_1, g_1(in,...,in),...,g_m(i_1,...,i_n)$

$ENDDO$

...

$ENDDO$

$ENDDO$

یک وابستگی از S_1 به S_2 وجود دارد اگر و تنها اگر مقادیر α, β وجود داشته باشند بطوریکه (1) α کوچکتر یا مساوی β باشد (Lexicographically)(2) دستگاه معادلات ذیل برقرار باشد.

$$F_i(\alpha) = g_i(\beta) \quad Forall \quad i \qquad 1 \le i \le m$$

برای بررسی این موضوع از علامت Δ استفاده می نماییم که اساس دلتا تست می‌باشد. در این حالت فرض می‌کنیم که به منبع در تکرار I_0 و مقصد در تکرار $I_0 + \Delta I$ دسترسی پیدا می کنیم به نوعی که ΔI تعداد تکرارها میان دو دسترسی (منبع و مقصد) می باشد.

مثال ذیل را فرض نمایید

$DO \quad I = 1, N$

$S \qquad A(I+1) = A(I) + B$

$ENDDO$

اگـر بخـواهیم یـک وابـستگی true داشـته باشـیم ،سـمت چـپ جملـه S بایـد در تکـرار I_0 بـه مکـان M از حافظـه دسترسـی پیـدا کـرده و سـمت راسـت جملـه S در تکـرار $I_0 + \Delta I$ بـه مکـان M دسترسـی پیـدا کنـد بنـابراین $A(I_0 + 1)$ و $A(I_0 + \Delta I)$ هـر دو بایـد بـه یـک مکـان حافظـه دسترسـی پیـدا کننـد. بنـابراین بایـد $I_0 + 1 = I_0 + \Delta I$ باشـد کـه نتیجـه دهـد ΔI برابـر ۱ مـی باشـد. از انجاییکـه ۱ بزرگتـر از صـفر مـی باشـد و کـوچکتر از حـد بـالا مـی باشـد (یعنـی حتمـا اجـرا مـی شـود) بنـابراین یـک وابـستگی true بـا مـسافت ۱ و بـردار(<) وجـود دارد. اگـر همـین تـست را بـرای ضـد وابـستگی انجـام مـی دادیـم آنگـاه بایـد ایـن شـرط برقـرار باشـد : $I_0 + 1 + \Delta I = I_0$ کـه نتیجـه مـی دهـد ۱- = ΔI بنابراین می توان گفت که چنین وابستگی وجود ندارد. مثال دیگری را در نظر بگیرید:

$DO \quad I = 1{,}100$

$\quad DO \quad J = 1{,}100$

$\quad\quad DO \quad K = 1{,}100$

$\quad\quad\quad A(I+1, J, K) = A(I+1, J, K+1) + B$

$\quad\quad ENDDO$

$\quad ENDDO$

$ENDDO$

$DO \quad I = 1{,}100$

$\quad DO \quad J = 1{,}100$

$\quad\quad DO \quad K = 1{,}100$

$\quad\quad\quad A(I+1, J, K) = A(I+1, J, K+1) + B$

$\quad\quad ENDDO$

$\quad ENDDO$

$ENDDO$

$I_0 + 1 = I_0 + \Delta I \quad ; \quad J_0 = J_0 + \Delta J \; ; K_0 = K_0 + \Delta K + 1$

$\Rightarrow \Delta I = 1 \; ; \; \Delta J = \phi \quad ; \; \Delta K = -1$

و بـردار جهـت آن $(<,=,>)$ مـی باشـد. اگـر انـدیس یکـی از حلقـه هـا در منبـع یـا مقـصد ظـاهر نـشود مـسافت آن قیـد خاصـی نـدارد.بـدین مفهـوم کـه مـی توانـد مـسافت معتبـری را بگیـرد. جهـت مربتط با آن را با × نشان می دهند که اجتماع هر سه جهت می باشد.

```
DO    I = 1,100
      DO    J = 1,100
            A(I+1) = A(I) + B(J)
            ENDDO
      ENDDO
```

بردار وابستگی آن $(*,<)$ می باشد. مثالی دیگر رادر نظر بگیرید:

```
DO    J = 1,100
      DO    I = 1,100
            A(I+1) = A(I) + B(J)
            ENDDO
      ENDDO
```

بـردار وابـستگی در ایـن مثـال $(>,*)$ مـی شـودکه خـود از سـه بـردار $[(=,<)(>,<)(<,<)]$ تـشکیل شـده اسـت. $(<,<)$ بـه معنـی یـک وابـستگی true سـطح ۱، $(=,<)$ بـه معنـی یـک وابـستگی true سـح ۲ مـی باشـد. $(>,<)$ بـدین مفهـوم نمـی باشـد کـه وابـستگی وجـود نـدارد بلکـه بـدین مفهـوم اسـت کـه منبـع مفهـوم و مقـصد وابـستگی اشـتباه گرفتـه شـده اسـت بنـابراین در جهـت معکـوس بـردار $(>,<)$ را خـواهیم داشـت کـه بـه مفهـوم یـک وابـستگی anti مـی باشـد. ایـن وابـستگی وجـود دارد بـرای مثـال در زمانی که I=2 , J=1 می باشد مقدار A(2) خوانده می شود که در I=1 , J=2 نوشته می شود.

۳. افزایش توازی

در ایـن بخـش تکنیـک‌هـایی بـرای مـوازی سـازی اتوماتیـک حلقـه هـا بـا اسـتفاده از تئـوری وابـستگی بررسـی مـی‌گـردد. میـزان بزرگـی انـدازه قطعـات مـوازی شـده در قالـب قطعـات بـا دانـه بنـدی نـازک یـا در اصـطلاح **Fine-Grained** و قطعـات بـا دانـه بنـدی درشـت **Coarse-Grained** مطـرح اسـت. در ایـن بخش تکنیکهایی برای موازی سازی با دانه بندی نازک مطرح می شود.

۳-۱ تعویض حلقه

تعـویض حلقـه یـا در اصـطلاح Loop Interchange بـدین مفهـوم مـی‌باشـد کـه ترتیـب حلقـه‌هـای تودرتوبه فرم دلخواه درآید.

```
      DO I=1, N
       Do J=1 , μ
S        A(I, J+1) = A (I, J) +B
      ENDDO
    ENDDO
```

در ایـن مثـال وابـستگی را حلقـه درونـی حمـل مـی‌کنـد بنـابراین Vectorization امکـان پـذیر نمـی‌باشـد. بنـابراین تعـویض حلقـه مـی توانـد ایـن مـشکل را حـل نمایـد زیـرا بـا تعـویض حلقـه در سـطح داخلی می توان موازی سازی نمود.

```
DO     J = 1, μ
       DO     I = 1, N
S                    A(I, J +1) = A(I, J) + B
       ENDDO
ENDDO
```

$$
\begin{array}{ll}
DO & J = 1, \mu \\
S & \quad A(1:N, J+1) = A(1:N, J) + B \\
 & ENDDO
\end{array}
$$

تعـویض حلقــه یکــی از تبــدیلات reordering محــسوب مــی شــود زیــرا کــه ترتیــب اجــرای دستورات را تغییر می دهد. همه تعویضات معتبر نیستند. برای نمونه به مثال ذیل توجه فرمایید.

$$
\begin{array}{ll}
DO & I = 1, \mu \\
 & \quad DO \quad J = 1, N \\
 & \qquad A(I, J+1) = A(I+1, J) + B \\
 & \quad ENDDO \\
 & ENDDO
\end{array}
$$

بــا تعــویض جــای دو حلقــه مقــصد وابــستگی زودتــر از منبــع وابــستگی اجــرا خواهــد شــد. همانگونــه کــه مــشاهده مــی شــود در قطعــه کــد اولیــه در مکــان A(2,2) در تکــرار I=2 ، J=1 نوشــته و در تکــرار I=1 و J=2 از آن مــی خوانــد زیــرا تکــرار I=2 ، J=1 قبــل از I=1 و J=2 اجــرا مــی شــود. ولــی در قطعــه کــه تغییــر یافتــه ابتــدا در تکــرار I=1 ، J=2 از مکــان A(2,2) مــی خــوانیم و در تکــرار I=2 ، J=1 در آن می نویسیم به عبارتی یک وابستگی داده در کد اولیه به یک وابستگی آنتی تبدیل شده است.

تعویض[۲۸] حلقه ها ممکن است با چند هدف صورت گیرد. مثلا در قطعه کد ذیل حلقه خــارجی نمــی توانــد به یک Do All موازی تبدیل شود و نیاز بــه همگــام ســازی بیــشتری دارد. امــا دو حلقــه مــی تواننــد تعــویض گردند .

```
      do I = 2, N
        do J = 3, M
S1:       A(I,J) = A(I-1,J-2) + C(I)*D(J)
        enddo
      enddo
```

[28] Interchanging

جابجایی حلقه ها در صورتی امکان پذیر است که هیچ وابستگی توسط حلقه خارجی تر حمل نشود و یک فاصله منفی با حلقه داخلی داشته باشد (بردار جهت (<,>) نداشته باشیم). در بردار فاصله، جهت وابستگی های حلقه های تعویض شده جای اجزای بردار یا دراصطلاح المانها عوض می شود. از آنجا که خارجی ترین حلقه با فاصله مثبت، حلقه خارجی تر j می باشد، حلقه j وابستگی را حمل می کند. در اینصورت حلقه i که وابستگی را حمل نمی کند می تواند بصورت موازی اجرا گردد. پس تعویض حلقه امکان اجرای حلقه های دیگر را می دهد.

```
      do J = 3, M
        do I = 2, N
S1:       A(I,J) = A(I-1,J-2) + C(I)*D(J)
        enddo
      enddo
```

تعریف ۱- یک وابستگی با توجه به یک جفت از حلقه ها، جفظ کننده عیر قابل تعویض یا در اصطلاح interchange Preventing می باشد اگر تغییر مکان آن حلقه ها نقطه پایانی وابستگی ها را دچار تغییر ترتیب recorder نماید.

تعریف۲- یک وابستگی حساس به تعویض یا در اصطلاح interchange sensitive می باشد اگر بعد از انجام عمل تعویض نیز توسط همان حلقه حمل شود. به عبارتی دیگر، یک وابستگی حساس به تعویض با حمل کننده اولیه خود به مکان جدید منتقل می گردد.

در یک وابستگی interchange Preventing ، بردار وابستگی پس از تعویض تغییر می کند که سبب می شود داده ها به ترتیب دیگر دسترسی شوند.

قضیه ۱- فرض کنید D(i,j) بردار جهت برای وابستگی در یک حلقه n تایی تو در تو باشد. بردار جهت برای همان وابستگی بعد از جایگشت حلقه ها به وسیله اعمال همان جایگشت به عناصر D(i,j) مشخص می گردد.

تعریف ۳- ماتریس جهت برای یک حلقه تو در تو ماتریسی است که هر سطر آن یک بردار جهت برای وابستگی موجود میان جملات آن می باشد.

$$
\begin{array}{l}
DO \quad I = 1, N \\
\quad DO \quad J = 1, \mu \\
\quad\quad DO \quad K = 1, L \\
\quad\quad\quad A(I+1, J+1, K) = A(I, J, K) + A(I, J+1, K+1) \\
\quad\quad ENDDO \\
\quad ENDDO \\
ENDDO
\end{array}
$$

ماتریس جهت در این حلقه $\begin{bmatrix} <,<,= \\ <,=,> \end{bmatrix}$ می باشد.

در این حلقه اگر حلقه خارجی به داخلی ترین موقعیت برود، ماتریس برای ان به شکل زیر در می آید: $\begin{bmatrix} < = > \\ = > < \end{bmatrix}$. سطر دوم، یک < به عنوان سمت چپ ترین عنصر نامساوی = پیدا کرده است که سبب می شود این جایگشت نامعتبر شود زیرا همان گونه که می دانیم این تغییر باعث جابجایی جهت وابستگی (منبع و مقصد وابستگی) می شود.

قضیه ۲- یک جایگشت از حلقه ها است اگر و تنها اگر ماتریس جهت پس از اعمال همان جایگشت به ستون هایش، در هیچ سطری < به عنوان سمت چپ ترین علامت که = نیست نداشته باشد.

۳-۲ مورب سازی حلقه

مورب سازی یا در اصطلاح Loop skewing تبدیلی می باشد که فضای تکرار را تغییر شکل می دهد تا بتوان پتانسیل توازی در یک فضای تکرار را با حلقه های موازی معمولی بیان نمود.

$$
\begin{array}{lll}
DO & I = 1, N & \\
 & DO \quad J = 1, N & \\
S & & A(I,J) = A(I-1,J) + A(I,J-1) \\
 & ENDDO & \\
 & ENDDO &
\end{array}
$$

در مثال بالا همان گونه که مشاهده می شود نقاطی که دارای قابلیت اجرای موازی می باشند در یک تکرار قرار نمی گیرند. یکی از روش های انجام این کار تغییر شکل دادن فضای تکرار می باشند به نوعی که نقاط قرار گرفته به روی یک قطر در یک تکرار انجام شود. یکی از نگاشت های ممکن ایجاد متغیر جدید اندیس j می باشد به نوعی که $j = J + I$ است. با نگاشت معکوس $J = j - I$ خواهد شد.

$$
\begin{array}{lll}
DO & I = 1, N & \\
 & DO \quad J = I + 1, I + N & \\
S & & A(I,J-I) = A(I-1,J-I) + A(I,J-I-1)
\end{array}
$$

همان گونه که مشاهده می شود برای این که همان نقاط آرایه A دسترسی شوند باید اندیس آرایه ها در حلقه تغییر نمایند.j=J+I در نتیجه J=j-I خواهد شد.

در حالت جدید نیز هر دو حلقه حامل وابستگی می باشند بنابراین موازی سازی امکان ندارد. از آنجائی که بردارهای وابستگی $(<,<)$ $(=,<)$ می باشد اگر جای دو حلقه عوض شود بردار وابستگی به صورت $(<,<)$ $(<,=)$ در می آید یا به عبارتی هر دو وایستکی توسط حلقه بیرونی حمل می شود.

$DO \quad J = 2, N + N$

$DO \quad I = \max(1, J - N), \min$

$A(I + J - I) = A(I - 1, J - I) + A(I, J - I - 1)$

محدوده جدید حلقه ها ناشی از فضای مثلثی تکرار می باشد. به شکل ۴ توجه نمایید.

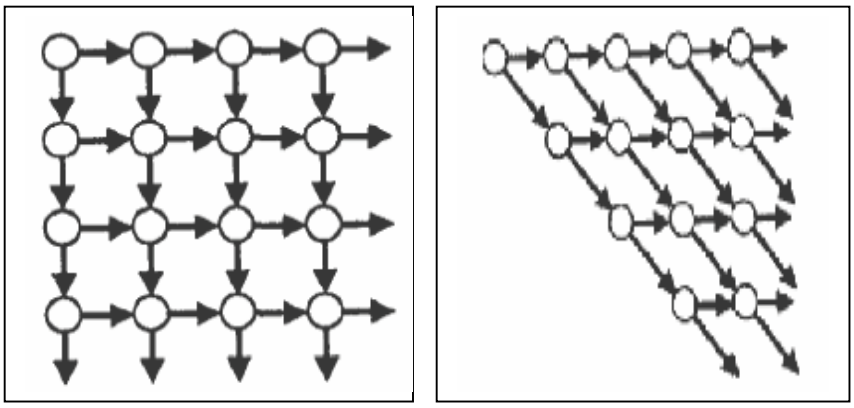
$DO \quad J = 2, N + N$

$ForALL(I = \max(1, J - N), \min(N, J - 1)$

$A(I, J - 1) = A(I - 1, J - I) + A(I, J - I - 1)$

$END \ \ FORALL$

$ENDDO$

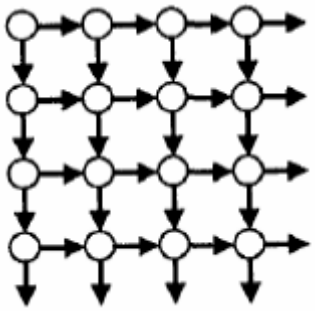


شکل ۴- فضاهی وابستگی مربعی و مثلثی

در برخی از حلقه های تودرتو، وابستگی ها با همه حلقه ها حمل می شود که مانع از اجرای موازی حلقه ها می گردد. برای نمونه قطعه کد ارایه شده در شکل ۵ توجه نمایید. بردارهای وابستگی برای اینحلقه عبارتند از:

S1 $\delta_{(0,1)}$ S1 S1 $\delta_{(1,0)}$ S1 S1 $\delta'_{(0,1)}$ S1 S1 $\delta'_{(1,0)}$ S1

```
      do I = 2, N-1
        do J = 2, M-1
S1:       A(I,J)=0.2*(A(I-1,J)+A(I,J-1)
                     +A(I,J)+A(I+1,J)+A(I,J+1))
        enddo
      enddo
```

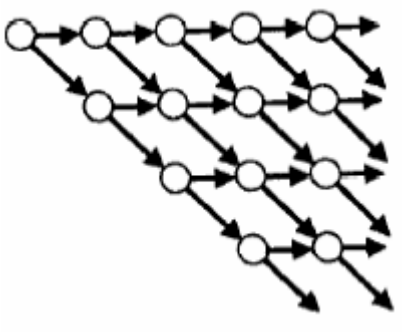


شکل ۵- یک حلقه و فضای وابستگی آن

برای تعیین تکرارهایی که به صورت موازی قابل اجرا هستند یک روش مورب سازی فضای تکرارهای حلقه است. بدین ترتیب کلیه تکرارهایی که در شکل هندسی فضای تکرار بر روی یک خط مورب قرار می گیرند در اصطلاح در جلوی یک موج یا wavefront قرار می گیرند در واقع با مورب سازی حلقه ها، شکل فضای تکرار مربعی[۲۹] به متوازی السطوح[۳۰] تغییر می کند. برای نمونه می توان حلقه J قطعه کد فوق را با اضافه کردن I به حدود پایین و بالای حلقه J مورب نمود. البته برای انجام این کار لازم است که در درون حلقه، I را از J کم کرد. حلقه مورب شده و فضای تکرار آن در شکل ٦ ارایه شده است.

```
      do I = 2, N-1
        do J = I+2, I+M-1
S1:       A(I,J-I)=0.2*(A(I-1,J-I)+A(I,J-I-1)
             +A(I,J-I)+A(I+1,J-I)+A(I,J-I+1))
        enddo
      enddo
```

[29] Rectangular
[30] Parallelogram



شکل ٦- حلقه مورب شده برای حلقه ارایه شده در شکل ٥

بردار جهت برای وابستگی ها در حلقه مورب شده از (d1,d2) به (d1,d1+d2) تغییر می کند پس وابستگی ها نیز به مقدار زیر تغییر می یابند:

S1 $\delta_{(0,1)}$ S1 S1 $\delta_{(1,1)}$ S1 S1 $\delta'_{(0,1)}$ S1 S1 $\delta'_{(1,1)}$ S1

برای تعویض جای حلقه های مورب شده، نیاز به تغییرات هوشمندانه ای در حدود حلقه است که در قطعه کد ذیل مشخص شده است.

```
      do J = 4, N+M-2
        do I = MAX(2,J-M+1), MIN(N-1,J-2)
S1:       A(I,J-I)=0.2*(A(I-1,J-I)+A(I,J-I-1)
             +A(I,J-I)+A(I+1,J-I)+A(I,J-I+1))
        enddo
      enddo
```

همانند قبل، تعویض جای حلقه ها نیازمند تعویض المانهای متناظر در بردارهای جهت می باشد.

S1 $\delta'_{(1,1)}$ S1 S1 $\delta'_{(1,0)}$ S1 S1 $\delta_{(1,1)}$ S1 S1 $\delta_{(1,0)}$ S1

نکته مهم اینکه در هر صورت مقدار مولفه اول بردار جهت مقدار مثبت است بدین معنا که هر وابستگی توسط حلقه خارجی تر (حلقه J) حمل می شود. لذا حلقه مورب و تعویض شده I می تواند موازی اجرا گردد.

۳-۲ نوار کشی[۳۱] حلقه ها

غالباً کارایی کد موازی فقط به میزان توازی که در آن یافت می شود بستگی ندارد بلکه به نحوه زمانبندی آن نیز بستگی دارد. برای زمانبندی بهینه نیاز به همگام نمودن وظایف موازی است. در اینجا عدم تساوی زمان اجرایی وظایف می تواند همگام سازی را دچار مشکل نماید. مسلماً هر چه تعداد وظایف همگام شونده کمتر باشد همگام سازی کمتری نیاز خواهد بود. ابزار ایجاد توازن میان موازی سازی و همگام سازی نوار بندی یا در اصطلاح Strip mining و خط لوله سازی یا در اصطلاح Pipelining می باشد که در این بخش توضیح داده خواهند شد.

بسیاری از حلقه های مستقل از وابستگی که تکرارهایشان می تواند به صورت موازی صحیح اجرا شوند، اگر زمان بندی های دقیق برای آنها به کار نرود اجرای موثری نخواهند داشت.

دسته بندی تکرارهای حلقه به صورت مجموعه هایی که واحد زمان بندی را تشکیل دهند می تواند موجب استفاده کارا از توازی گردد. این نوع تبدیل برای برداری کردن، نوار بندی یا در اصطلاح Strip mining می باشد. مثال ذیل استفاده از این نوع تبدیل را برای موازی سازی نشان می دهد.

$$
\begin{aligned}
&DO\ I = 1, N \\
&\qquad A(I) = A(I) + B(I) \\
&ENDDO
\end{aligned}
$$

[31] Strip Mining

اگر دقیقاً P پردازنده برای اجرای حلقه موجود باشد بهترین توزیع بار و همزمان سازی به صورت ذیل امکان پذیر است.

$$K = CEIL(N/P)$$
$$PARALLEL \quad DO \quad I = 1, N, K$$
$$Do\ j = 1, MIN(I + K - 1, N)$$
$$Y = 1 + j - 1$$
$$A(y) = A(y) + B(j)$$
$$ENDDO$$
$$END \quad PARALLEL \quad DO$$

در حلقه‌هایی تو در تو مانند حلقه فوق که همگی حلقه‌های داخلی به یک میزان محاسبات نیاز دارند، پیدا کردن توازن دقیق زمانی که تعداد پردازنده‌ها و تکرارهای حلقه‌ها مشخص باشد آسان می‌باشد. از آنجائی که این مقادیر تا قبل از زمان اجرا معمولاً نامشخص هستند، نوار بندی را به وسیله سخت افزار خاص انجام می‌دهند.

در مواردی که فاصله زمان اجرا میان تکرارهای مختلف متفاوت می‌باشد فراهم‌سازی یک توازن موثر بسیار مشکل می‌باشد و به جای این که حلقه خارجی میان مقسوم‌علیه‌ای از پردازنده‌ها تقسیم شود، انتخاب یک اندازه بلاک کوچکتر مناسب‌تر می‌باشد. بدین وسیله، پردازنده‌هایی که کار کمتری انجام داده و کارشانزودتر پایان می‌پذیرد، می‌توانند در حالی که پردازنده‌های با بارکاری بیشتر مشغول کار هستند،تعدادی از کارهای باقیمانده را انجام دهند. اندازه بلاک کوچکتر این فایده را نیز دارد که به پردازنده‌هایی که زودتر کار خود را شروع کرده‌اند اجازه می‌دهد بار بیشتری را به عهده گیرند.

کامپایلرهای برداری اغلب یک حلقه را به یک جفت حلقه تبدیل می کنند که حداکثر تعداد دفعات تکرار حلقه داخلی برابر با حداکثر طول بردار ماشین است. بعنوان مثال قطعه کد ذیل را در نظر بگیرید

```
      do I = 1, N
S1:     A(I) = A(I) + B(I)
S2:     C(I) = A(I-1) * 2
      enddo
```

در یک ماشین برداری Cray حلقه فوق به جفت حلقه های ذیل تبدیل خواهد شد. به این نوع تبدیل در اصطلاح لفظ نوار کشی یا Strip mining اتلاق می شود.

```
      do IS = 1, N, 64
        do I = IS, MIN(N,IS+63)
S1:       A(I) = A(I) + B(I)
S2:       C(I) = A(I-1) * 2
        enddo
      enddo
```

در اینجاحلقه اصلی به نوار هایی با اندازه ماکزیمم بنام اندازه نوار[۳۲] تقسیم می شود. در قطعه کد فوق، حلقه داخلی تر (بنام حلقه پایه[۳۳]) دارای اندازه نوار ٦٤ می باشد که در واقع همان طول ثباتهای بردار Cray است. حلقه خارجی تریعنی حلقه IS (بنام حلقه نوار[۳٤]) بین نوارها گام برمی دارد. (در ماشین Cray حلقه I مربوط به دستورات برداری[۳۵] می باشد).

تبدیل نوار کشی همواره مجاز است حتی اگر تاثیری در رابطه وابستگی داده ها در حلقه داشته باشد. از آنجا که انجام نوارکشی حلقه ای را اضافه می کند، در واقع یک بعد به فضای تکرار اضافه خواهد شد پس باید یک المان نیز به بردار فاصله یا به بردار جهت نیز اضافه گردد. اگر فاصله وابستگی حلقه اصلی بزرگتر

[32] Strip Size
[33] Element Loop
[34] Strip Loop
[35] Vector Instruction

از (یا مساوی با) اندازه نوار باشد، پس از انجام نوارکشی، حلقه نوار وابستگی را حمل خواهد کرد و لذا امکان اجرای موازی حلقه پایه را می دهد.

مثال ذیل شکلی از حلقه که مناسب برای خط لوله سازی است را نشان می دهد. در این مثال نمونه‌های محتلف جمله S_1 می‌توانند به صورت موازی مانند یک حلقه موازی انجام شوند اما، نمونه‌های S_2 باید منتظر تکرار قبلی خود بمانند. در این مثال POST (EV(I)) انجام رویداد EV(I) را اعلام می‌نماید و WAIT (EV(I)) تا هنگام رسیدن رویداد بلوکه می‌شود.

$$DOACROSS \quad I = 2, N$$
$$A(I) = B(I) + C(I)$$
$$S1 \qquad POST(EV(I))$$
$$If \ \ (I.GT.2) \ WAIT \ EV(I-1)1$$
$$S_2 \qquad C(I) = A(I-1) + A(I)$$
$$ENDDO$$

در ادامه مثالی که موضوع را واضح‌تر می‌نماید ذکر قبل شده است. این مثال قبلا با استفاده از تبدیل مورب سازی موزای شده بود.

$$DO \qquad I = 2, N-1$$
$$DO \qquad J = 2, N-1$$
$$A(I,J) = 0.25 * (A(I-1, j), A(I, J-1)8$$
$$+ A(I+1, J) + A(I, J, 1)$$

نکته مهم در اینجا این می باشد که A(I+1,J) قبل از A(I,J) محاسبه نگردد. در این مثال نیز همانند مثال قبل از آرایه‌ای از رویدادها به منظور اطمینان از درستی محاسبات استفاده می‌شود.

$$DOACROSS \quad I = 2, N-1$$
$$POST(EV(1))$$
$$DOJ = 2, N-1$$

$WAIT(EV(J-1))$

$A(I,J) = 0.25 * (A(I-1,J) + A(I,J-1) + A(I+1,J) + A(I,J+1))$

$POST(EV(J))$

$ENDDO$

$ENDDO$

اگـر سـربار قابـل توجـه‌ای در سـنکرون سـازی باشـد، ممکـن اسـت تکرارهـا بـه منظـور کـاهش فرکانس سنکرون‌سازی گروه‌بندی شوند.

$DOACROSS \quad I = 2, N-1$

$POST(EV(1))$

$K = \phi$

$DO \quad J = 2, N-1,2$

$K = K+1$

$WAIT(EV(K))$

$DO\, m = J, \mu A \times (J+1, N-1)$

$A(I,m) = 0.25 * (A(I-1,m) + A(I,m-1)8$

$+ A(I+1,m) + A(I,m+1)$

$ENDDO$

$POST(EV(K+1))$

$ENDDO$

$ENDDO$

در اینجـا بـرای کـاهش سـربار سـنکرون سـازی درجـه مـوازی سـازی کـاهش یافتـه اسـت بـه دلیـل ایـن کـه ارزش DOACROSS بـسیار وابـسته بـه ماشـین مـی‌باشـد. تنهـا زمـانی بـه کـار مـی‌رود کـه تنهـا راه افزایش کارایی باشد.

۴. کاشی کاری فضای تکرار

هدف از کاشی بندی، دانه بندی مناسب تکرارهای حلقه های ترتیبی تو در تو جهت توزیع بهینه تکرارها در بین پردازنده ها است به طوری که هزینه ارتباط ها و همگام سازی بین پردازنده ها کمینه گردد. برای این منظور، فضای یکنواخت وابستگی تکرارهای حلقه به قطعاتی به نام کاشی با حداقل وابستگی افراز می شود [۳۲، ٤٤ و ٦٦]. کاشی بندی یک نوع افراز بندی یا خوشه بندی در فضای دکارتی است. هدف ا یافتن شکل و اندازه کاشی مناسب جهت موازی سازی حلقه هاست [٤٧، ۷۱ و ۷۲] به طوری که: (۱) زنجیره ای از بردارهای وابستگی بین دو تکرار وابسته در داخل یک کاشی، در داخل آن کاشی باشند، (۲) تعداد بردارهای وابستگی بین یک کاشی با کاشی های همسایه، کمینه باشد و (۳) هر محدودیتی که به وسیله کاربر اعمال می شود، بر آورده شود.

یکی از اولین تلاشها در زمینه کاشی بندی فضای تکرار حلقه ها در اواسط دهه ۱۹۸۰ و توسط وولف به ثمر رسید [۷۸، ۷۹، ۸۰، ۸۱ و ۸۲]. در روش وولف فضای تکرار حلقه ها با استفاده از تکنیک مورب سازی به یک فضای متوازی السطوح تبدیل شده و سپس کاشی بندی می شود. عمده مشکل این رویکرد پیدا کردن یک تبدیل بهینه جهت مورب سازی حلقه است به طوری که تعداد دسترسی ها به حافظه برای اجرای تکرارها بهینه گردد. پیدا کردن این تبدیل با افزایش عمق حلقه های تو در تو افزایش می یابد. برای این منظور وولف یک الگوریتم مکاشفه ای را برای حل این مشکل ارایه داده است. هم چنین این مسئله به وسیله قریل و همکارانش [۳۳] نیز مطرح شده است. این رویکرد به عنوان یک روش بنیانی مبنایی برای تحقیقهای بعدی در این زمینه ها مورد استفاده واقع شده است.

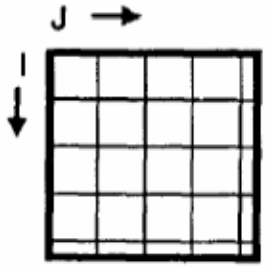
با نوار کشی حلقه های تودرتو و جابجایی حلقه های نوار به خارجی ترین سطح فضای تکرار کاشی کاری خواهد شد. برای نمونه قطعه کد ذیل که شامل دو حلقه تودرتو است را در نظر بگیرید:

```
      do I = 1, N
        do J = 1, N
S1:       A(I,J) = A(I,J) + B(I,J)
S2:       C(I,J) = A(I-1,J) * 2
        enddo
      enddo
```

پس از نوار بندی این حلقه تبدیل به چهار حلقه تودرتوی ذیل می گردد:

```
      do IT = 1, N, SS
        do JT = 1, N, SS
          do I = IT, MIN(N, IT+SS-1)
            do J = JT, MIN(N, JT+SS-1)
S1:           A(I,J) = A(I,J) + B(I,J)
S2:           C(I,J) = A(I-1,J) * 2
            enddo
          enddo
        enddo
      enddo
```

بعبارتی دیگر فضای دو بعدی تکرار برنامه (۱۵ – الف) به کاشی هایی که در شکل ۵ نمایش داده شده است تقسیم می شوند. هر کاشی مربوط به دو حلقه پایه داخلی است و دو حلقه خارجی تر بنام کاشی، بین کاشی ها گام برمی دارند.



شکل ۵- فضای کاشی کاری شده

کاشی کاری فضای تکرار نامنظم بسیار پیچیده تر از فقط نوار کشی کردن هر حلقه خواهد بود. برای نمونه حلقه ذیل با فضای تکرار مثلثی را در نظر بگیرید:

```
do I = 1, N
  do J = I, N
    ...
```

کاشی کاری توسط نوارکشی مستقل هر حلقه، فضای تکرار مناسبی را در اینجا ایجاد نمی کند. برای داشتن فضای تکراری مناسب است باید حلقه ها مانند به صورت ذیل نوارکشی شوند:

```
do IT = 1, N, SS
  do JT = IT, N, SS
    do I = IT, MIN(N, IT+SS-1)
      do J = MAX(JT,I), MIN(N, JT+SS-1)
        ...
```

به عنوان یک نمونه قطعه کد ذیل را در نظر بگیرید:

for $j_1 = 0$ **to** 9 **do**
  for $j_2 = 0$ **to** 8 **do**
    a $(-4j_1 - j_2, -j_1 - 3j_2 + 3) = \ldots$
    $\ldots = \ldots$ a $(-j_1 + 1, -j_2 + 4)$ …

دستگاه معادلات و نامعادلات حاصل از مساوی قرار دادن اندیس آرایه ها در دو تکرار مختلف به صورت ذیل است:

$$-4j_1 - j_2 = -j'_1 + 1$$
$$-j_1 - 3j_2 + 3 = -j'_2 + 4$$

دستگاه معادلات پس از انتقال مقادیر ثابت به یک سمت تساوی به صورت ذیل خواهد بود:

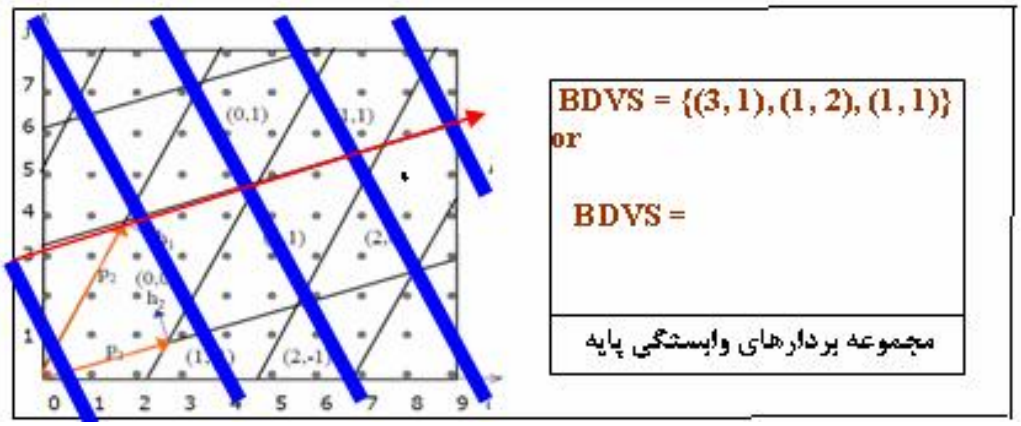
$$-4j_1 - j_2 + j'_1 = 1$$
$$-j_1 - 3j_2 + j'_2 = 1$$

پس از حل دستگاه معدلات به روش دیوفانتین تکرارهای وابسته را می توان مشخص نمود. البته مشکل در اینجا نگهداری شماره تکرارهای وابسته است. برای این منظور می توان بر اساس معادلات پارامتریک

حاصل از روش دیوفانتین بردارهای اولیه که بر مبنای آنها می توان تکرارهای وابسته را مشخص نمودتعیین کرد. جهت سادگی این بردارهای اولیه به عنوان بردارهای وابستگی در نظر گرفته می شوند و به این ترتیب فضای وابستگی به یک فضای یکنواخت تبدیل می شود. بردارهای اولیه برای قطعه کد فوق به صورت ذیل است:

BDVS = {(3, 1), (1, 2), (1, 1)}

برای کاشی بندی فضای تکرار همانگونه که در بالا توضیح داده شد می توان از روش مورب سازی حلقه ها و نواربندی استفاده نمود. برای نمونه فضای کاشی بندی شده برای حلقه فوق به صورت ذیل است:



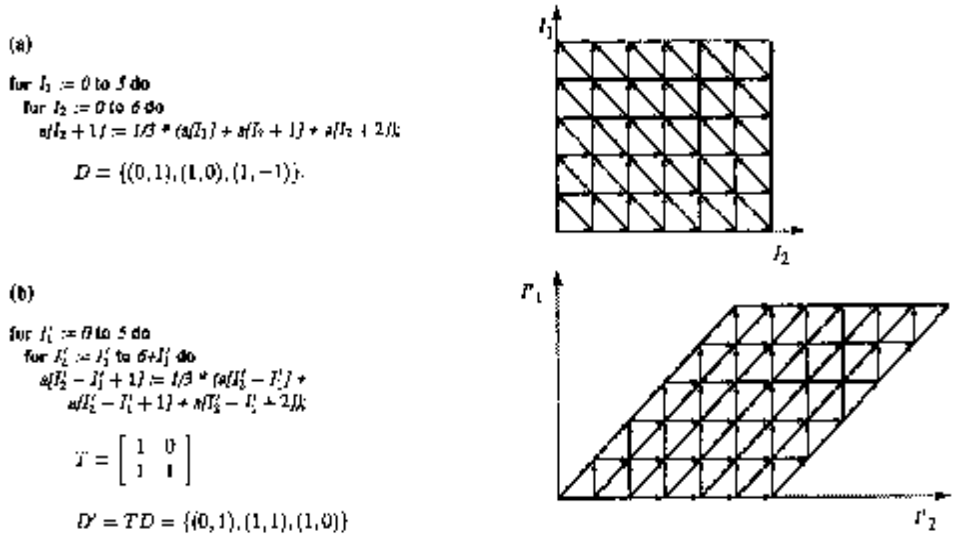
شکل ۶ – فضای یکنواخت کاشی بندی شده

در روش وولف، فضای یکنواخت تکرارهای حلقه با استفاده از یک تبدیل، به فضای مورب مناسب برای کاشی بندی تبدیل می شود. با استفاده از این تبدیل فضای وابستگی بین تکرارها تبدیل به فضایی مثبت با بردارهای وابستگی در جهت مثبت می شود.. برای نمونه به شکل ۷ توجه نمایید. در این شکل فضای تکرارها شامل ٦×۷ نقطه میباشد. در واقع هر نقطه نشانگر یک تکرار حلقه است. در این شکل $\vec{d}$

بردار وابستگی بین تکرارهای حلقه می باشد. در حالت کلی چنانچه تکرار $\vec{P}_1$ قبل از $\vec{P}_2$ می بایست به اجرا در آید در صورتیکه بردار فاصله $\vec{d}$ وجود داشته باشد به قسمیکه $\vec{P}_2 = \vec{P}_1 + \vec{d}$. در واقع بردار های وابستگی ترتیب اجرا تکرارها را مشخص می کنند. بردار ی در صورتی از لحاظ الفبایی مثبت است $\vec{d} > 0$ در صورتیکه:

$\exists\, i : d_i > 0 \text{ and } \forall\, j < i \; : d_j \geq 0$

در شکل ۷.a خانه های آرایه a در هر تکرار حلقه مقدار دهی می شوند. مقدار $a[I_2]$ از تکرار قبلی در حلقه داخلی حاصل می گردد. در صورتیکه دو دسترسی دیگر یعنی $a[i_1+1]$ و $a[i_1+2]$ هر دو وابسته به تکرارهای حلقه خارجی هستند. لبه های وابستگی در اینجا همگی از لحاظ الفبایی مثبت و به صورت مجموعه $\vec{d} = \{(۱-، ۰)، (۰،۱) ، (۱،۰)\}$ مشخص شده است.



شکل۷- نمونه ای از بکارگیری روش وولف برای کاشی بندی فضای تکرار یک حلقه

در قسمت بالای شکل ۷ یک حلقه و فضای تکرار آن مشخص شده است. برای اینکه بتوان فظای تکرارهای حلقه را کاشی یندی کرد به قسمی که بتوان کاشی های مستقلی را بدست آورد که بتوانند بطور موازی با یکدیگر به اجرا در آیند از تبدیلاتی مثل تعویض حلقه ها یا در اصطلاح loop interchange، مورب سازی skewing و برگشت reversal استفاده می شود. این نوع تبدیلات را می توان توسط ماتریسهای تبدیل اعمال نمود. تعویض حلقه ها برای نمونه تکرار (j, i) را با (i, j) جایگزین می کند. برای این منظور می توان از فرم ماتریسی به صورت ذیل استفاده نمود:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} j \\ i \end{bmatrix}$$

از آنجاییکه یک تبدیل یوتیماجولار تبدیلی خط است بنابراین:

$$T\vec{P}_2 - T\vec{P}_1 = T(\vec{P}_2 - \vec{P}_1) = T\vec{d}$$

بنابراین اگر در فضای اولیه $\vec{d}$ بردار فاصله باشد آنگاه در فضای تبدیل شده $T\vec{d}$ نماینگر بردار فاصله خواهد بود. بنابر این در اینجا بردار $\vec{d} = (d_1 ، d_2)$ به صورت ذیل تبدیل می شود:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_1 \end{bmatrix}$$

سه تبدیل اولیه در حلقه ها وجود دارد. که عبارتند از:

- **جابجایی**: جابجایی یا در اصطلاح پرمیوتیشن σ در یک حلقه تکرار $(P_1, P_2, \ldots P_n)$ را به تکرار $(P_{\sigma 1}, P_{\sigma 2}, .. P_{\sigma n})$ می نماید. برای این منظور می بایست سطرهای ماتریس $n \times n$ یکه I را به اندازه σ جابجا نمود تا ماتریس تبدیل I_σ بدست آید. با استفاده از ماتریس تیدیل بدست آمده می توان عمل جابجایی σ را در فضای تکرارهای حلقه مورد نظر ایجاد نمود.

- **برگشت:** برگشت یا ریورسال i امین حلقه با استفاده از ماتریس یکه که عنصر i ام بر روی قطر آن مساوی با ۱- است حاصل می گردد.
- **مورب سازی**: مورب سازی حلقه I_j نسبت به حلقه I_i به اندازه فاکتور صحیح f تکرار:

$(P_1, P_2, \ldots, P_{i-1}, P_i, P_{i+1}, \ldots, P_{j-1}, P_j, P_{j+1}, \ldots P_n)$

به صورت ذیل تبدیل می کند:

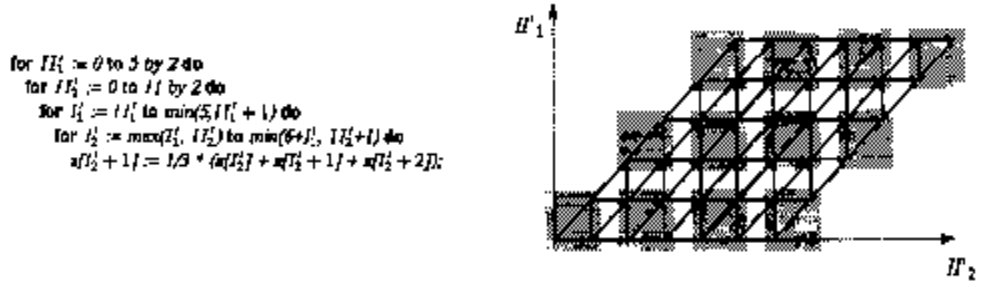
$(P_1, P_2, \ldots, P_{i-1}, P_i, P_{i+1}, \ldots, P_{j-1}, P_j + f P_i , P_{j+1}, \ldots P_n)$

ماتریس تبدیل T برای عمل مورب سازی با این تغییر در ماتریس یکه بدست می آید که عنصر $t_{j,i}$ در ابن ماتریس بجای صفر مساوی با مقدار f قرار داده می شود. از آنجاییکه $i < j$ است ماتریس تبدیل T می بایست زیر مثلثی باشد. به عنوان نمونه جهت مورب سازی فضای تکرار شکل ۷.a به شکل ۷.b تبدیل شود، ماتریس تبدیل ذیل لازم می باشد

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

کلیه این تبدیلات توسط ماتریسهای یونیماجولار انجام می شود. یک ماتریس یونیماجولار دارای سه خاصیت مهم می باشد. اولا"، مربعی است به این مفهوم که یک فضای تکرار n بعدی را به فضای تکرار n بعدی تصویر می نماید. ثانیا"، کلیه عناضر را در نظر می گیرد و هیچ عنصری را نادیده نمی گیرد. بنابراین یک وکتور از شمارنده های حلقه که اعداد صحیح می باشند را به وکتوری از اعداد صحیح تبدیل می کند. ثالثا" قدر مطلق دترمینان آن یک می باشد. بخاطر این سه ویژگی حاصل ضرب و معکوس یک ماتریس یونیماجولار نیز یونیماجولار می باشد. می توان تبدیلات ترکیبی را از دنباله ای از تبدلات حاصل نمود.

پس از اعمال الگوریتم وولف فضای تکرار مورب شده و حلقه مطابق شکل 7.b خواهد بود. بعد از اعمال تبدیل های لازم بر روی فضای یکنواخت تکرارهای وابسته حلقه نوبت به کاشی بندی جهت اجرای موازی تکرارهای حلقه در قالب دانه بندی درشت می رسد. شکل ۸ نمایانگر فضای کاشی بندی شده و کد موازی حاصل از پیمایش فضای کاشی بندی شده است. در کد ارایه شده در شکل ۸، دو حلقه داخلی سبب اجرای ترتیبی نقاط داخل کاشی ها می شود و دو حلقه بیرونی سبب اجرای کاشی ها بر روی پردازنده ها می شود.



شکل۸- فضای کاشی بندی شده حاصل از رویکرد وولف و کد موازی حاصل از پیمایش کاشی ها

۵. تولید کد موازی

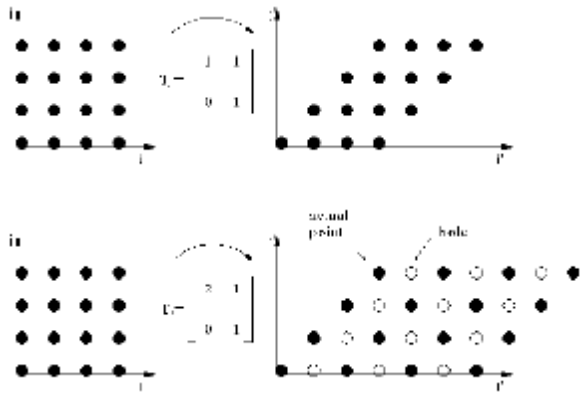
برای تولید کد موازی می توان فضای کاشی بندی شده را پیمایش نمود. نکته اصلی بدست آوردن تعداد ماکزیمم کاشی هایی است که به صورت موازی در هر زمان قابل اجرا هستند. کاشی هایی را که به صورت همزمان می توانند به اجرا در آیند ابتدا بر روی خطوطی به نام جلوی موج یا در اصطلاح Wavefront

مشخص می کنند [۳، ۲۵، ۳۲، ۵۲، ٦٦ و ۹۲]. برای نمونه به موجهای مشخص شده در شکل ٦ توجه نمایید بر اساس این خطوط امواج می توان کد موازی برای حلقه مربوطه را به صورت ذیل ایجاد نمود:

```
For WaveFrontNumber := -1 To 3 Do
  ForAll j^s_1 := -2 To 3 Do
    ForAll j^s_2 := -1 To 2 Do
      If j^s_1 + j^s_2 := WaveFrontNumber Then
        For j_1 := max (0, 3j^s_1 + 2j^s_2) To min (9, 3j^s_1 + 2j^s_2 - 4) Do
          For j_2 := max (0, j^s_1 + 4j^s_2) To min (8, j^s_1 + 4j^s_2 - 4) Do
            If (⌊HJ′⌋)′= J^s Then
              a[- 4j_1 - j_2, - j_1 - 3j_2 + 3] := …
              … := … a[- j_1 + 1, - j_2 + 4] …
            EndIf
          EndFor
        EndFor
      EndIf
    EndForAll
  EndForAll
EndFor
```

در روش ارایه شده توسط گوماس و آتاناسکی [۳۱] بعد از کاشی بندی فضای تکرار حلقه ها، کد موازی برای فضای تکرار کاشی شده با استفاده از روش فوریه–موتخین ایجاد می شود. در این روش سعی بر آن است که حدود حلقه های کد جدید به صورت یک عبارت محاسباتی بر اساس روش فوریه–موتخین محاسبه گردد. برای این منظور به جای استفاده از تبدیل های یک ریخت که کار تولید کد را ساده می کنند، از تبدیل های غیر یک ریخت استفاده می شود. تولید کد برای تبدیل غیر یک ریخت مشکل تر از تبدیل یک ریخت می باشد. در این رویکرد برای تولید کد موازی از همان ماتریس تبدیلی استفاده

می شود که برای نمایش یک کاشی به کار برده می شود. شکل ۹ نمونه هایی از نحوه تبدیل های یک ریخت و غیر یک ریخت را نشان می دهد.



شکل ۹- استفاده از تبدیل های یک ریخت و غیر یک ریخت در [۳۱]

۶. مثالی از موازی سازی حلقه

برای بدست آوردن تکرارهای وابسته در حلقه های تو در تو می توان از روش دیوفانیت استفاده نمود. در حالت کلی دستگاه معادلات ذیل که در آن ضرایب همگی اعداد صحیح هستند را در نظر بگیرید:

$$a_{11}x_1 + a_{21}x_2 + \ldots + a_{m1}x_m = c_1$$
$$a_{12}x_1 + a_{22}x_2 + \ldots + a_{m2}x_m = c_2$$
$$\ldots$$
$$a_{1n}x_1 + a_{2n}x_2 + \ldots + a_{mn}x_m = c_n$$

در دستگاه معادلات فوق تمام ضرایب a_{ij} و ضرایب ci اعداد صحیح هستند. فرض کنید U یک ماتریس Uniodular با ابعاد m×m و S ماتریس اخلون m×n است به قسمی که UA = S آنگاه دستگاه معادلات XA = C دارای جواب است اگر و تنها فقط اگر برداری از اعداد صحیح مثل T وجود داشته باشد به قسمی که TS = C باشد. هنگامی که جوابی موجود باشد آنگاه مجموعه جوابها از رابطه X = TU بدست می آید. در این حالت T یک بردار از اعداد صحیح است و در رابطه TS = C صدق می نماید زیرا:

C = XA = TUA = TS

در صورت وجود همچنین T ای آنگاه کلیه جوابها فرم X= TU را خواهند داشت. برای نمونه حلقه ذیل را در نظر بگیرید:

```
/* Original Sequential Code: */
   for j1 = 0 to 5
      for j2 = 0 to 5
S:       a (-j2, -j1 + 3) = …
```

T: … = … a $(-2j_2 + 1, j_1 + 4)$ …

end

end

با مساوی قرار دادن ضرایب رابطه ذیل دستگاه معادلات خطی به شرح ذیل حاصل می گردد:

$$-j_2 = -j'_2 + 1 \qquad -j_2 + j'_2 = 1$$
$$-2j_1 + 3 = -j'_1 + 4 \quad \Rightarrow \quad -2j_1 + j'_1 = 1$$

می توان با در نظر گرفتن دستگاه معادلات فوق رابطه ذیل را ایجاد نمود:

$$[j_1\ j_2\ j'_1\ j'_2] \begin{pmatrix} 0 & -2 \\ -1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} = [1\ 1]$$

با تبدیل ماتریس A به فرم اخلون ماتریسهای U و S حاصل می گردند که در واقع ماتریس U شاخص تغییراتی است که جهت تبدیل A به S بر روی ماتریس یکه انجام شده است. در واقع U یک ماتریس Unimodular و S ماتریس اخلون است. در ادامه این دو ماتریس مشخص شده اند:

$$U = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \text{ and } S = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

در این صورت راه حل عمومی برای مساله $[t_1\ t_2\ t_3\ t_4] \times S = [1\ 1]$ به صورت خواهد بود. به این ترتیب یک پاسخ صحیح برای دستگاه معادلات خطی به صورت ذیل وجود دارد.

$$X = TU = [1\ 1\ t_3\ t_4] \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{pmatrix} = [t_3\ \ t_4\ \ 2t_3 + 1\ \ t_4 + 1]$$

بردارهای وابستگی به صورت ذیل خواهند بود:

$$D = (d_1, d_2) = (j'_1 - j_1, j'_2 - j_2) = (t_3 + 1, 1)$$

در فرم ماتریسی می توان بردار D را به صورت ذیل مشخص نمود:

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} t_3 \\ t_4 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

در مثال فوق BDVS یا مجموعه بردارهای پایه وابستگی {(۱،۱) ،(۱،۰)} است زیرا، بر طبق رابطه فوق بردار وابستگی $D = t_3 \times (1, 0) + (1, 1)$ است. فضای وابستگی کاشی بندی شده و موجها برای آن در شکل ۸ مشخص شده است.

BDVS =

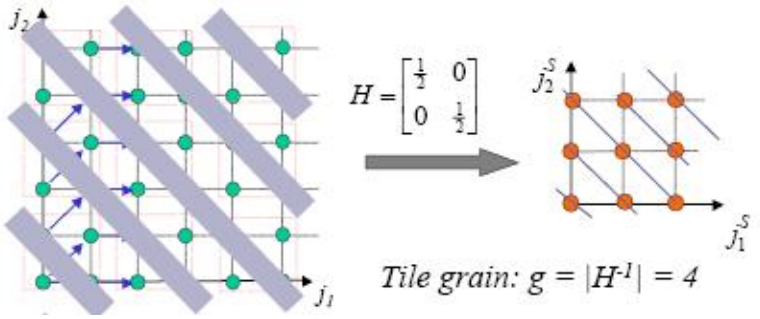
$UB = (5, 5) \Rightarrow LB = (0, 0), UB^s = (2, 2)$

$J^2 = \{J = (j_1, j_2) \mid 0 \le j_1, j_2 \le 5\} \Rightarrow J^{s2} = \{J^s = (j^s_1, j^s_2) \mid 0 \le j^s_1, j^s_2 \le 2\}$

With using the proposed tiling algorithm, we get the following tiling matrix:

$$P = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, H = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$$

$$| J^2 | = (ub_1 + 1) \times (ub_2 + 1) = (5 + 1) \times (5 + 1) = 6 \times 6 = 36 \Rightarrow | J^{s2} | = | J^2 | / |P_{2\times2}| = 36 / 4 = 9$$



شکل ۸- فضای کاشی بندی شده

همانگونه که در شکل فوق مشخص است، تعداد کاشی ها ۹ عدد و تعداد لبه های موج ۵ عدد می باشد. ابه این ترتیب میزان توازی می تواند تا ۹/۵ بدست آید. حلقه موازی با در نظر گرفتن شکل ۸ به صورت ذیل برای حلقه فوق حاصل می گردد:

Parallel forms:
1)
for w = 0 **to** 4 **in sequential**
  for j^s_1 = 0 **to** 2 **in parallel**
    for j^s_2 = 0 **to** 2 **in parallel**
      if $j^s_1 + j^s_2$ = w **then**
        for $j_1 = 2j^s_1$ **to** $2j^s_1 + 1$ **in sequential**
          for $j_2 = 2j^s_2$ **to** $2j^s_2 + 1$ **in sequential**
S: a $(-j_2, -j_1 + 3)$ = …
T: … = … a $(-2j_2 + 1, j_1 + 4)$ …

2)
for w = 0 **to** 4 **in sequential**
  for j^s_1 = max(0, w - 2) **to** min (w , 2) **in parallel**

```
for j^s_2 = max( 0, w - 2) to min (w, 2) in parallel
    if j^s_1 + j^s_2 = w then
        for j_1 = 2j^s_1 to 2j^s_1 + 1 in sequential
            for j_2 = 2j^s_2 to 2j^s_2 + 1 in sequential
S:              a (-j_2, -j_1 + 3) = …
T:              … = … a (-2j_2 + 1, j_1 + 4) …
```

3)

```
for w = 0 to 4 in sequential
    for j^s_1 = min (w , 2), j^s_2 = max( 0, w - 2) to max( 0, w - 2), min (w , 2) step -1, +1 in parallel
        for j_1 = 2j^s_1 to 2j^s_1 + 1 in sequential
            for j_2 = 2j^s_2 to 2j^s_2 + 1 in sequential
S:              a (-j_2, -j_1 + 3) = …
T:              … = … a (-2j_2 + 1, j_1 + 4) …
```

۷. زمانبندی

همانگونه که در بالا توضیح داده شد مساله تشخیص توازی در کد ترتیبی به تعیین گراف وظایف منتهی شد. تشخیص وابستگی در بین تکرارهای حلقه به مساله تشخیص لبه های امواج بر روی فضای کاشی بندی شده تکرارهای حلقه خاتمه یافت. زمانبندی بهینه گراف وظایف برای اجرای موازی یکی از مسائل عمده در پردازش موازی می باشد.هدف از زمانبندی تخصیص بهینه وظایف به پردازنده های موازی است به قسمی که برنامه مورد نظر در حداقل زمان لازم به اجرا در می آید. زمانبندی در چنین روشی به این صورت انجام می‌شود که تقدم در میان وظایف برنامه حفظ می‌شوند . زمان اتمام کل یک برنامه موازی عموما طول برنامه یا زمان شروع تا پایان خوانده می‌شود بعضی از الگوریتم‌های دیگر تلاش بر کاهش هزینه‌های پردازشگرهای موازی دارند.

دو روش ایستا و پویا در این راستا مطرح است. نکته قابل توجه NP-Complete بودن مساله زمانبندی گراف وظایف است. برای این منظور الگوریتمهای مکاشفه ای و روشهای قطعی مطرح شده اند. یکی از روشهای ایستا و قطعی مطرح الگوریتم MCP است که برای گرافهای نسبتا" کوچک در عمل پاسخهای بهینه ایجاد نموده است.

الگوریتم اصلی MCP : یک DAG شامل مجموعه گره های $\{n_1, n_2, \ldots, n_n\}$ است که توسط یالهای $e(n_i, n_j)$ به همدیگر متصل شده اند. هر گره نشان دهندهٔ یک Task می- باشدو وزن گره یعنی $W_n(n_i)$ زمان اجرای Task است . هر یال نشان دهندهٔ پیغام ارسالی از گره ها به یکدیگر است و وزن یال یعنی $W_l(n_i, n_j)$ برابر با زمان انتقال پیغام است . زمانیکه دو گره به یک پردازندهٔ واحد زمانبندی

گردند ، وزن یال اتصالی آنها صفر خواهد بود. هر DAG دو گره با وزن صفر وجود دارد. گره شروع که شروع برنامه را نشان می- دهد و گره پایانی یا خروجی که مشخص کنندهٔ پایان برنامه است. قبل از هر چیزیبه شرح چند اصطلاح باید پرداخت:

- $T_{flevel}(n_i)$: طول طویلترین مسیر از گره n_i به گره پایانی است که وزن گره n_i را شامل آن نیست.
- $T_{blevel}(n_i)$: : طول طویلترین مسیر از گره n_i به گره پایانی است که وزن گره n_i را نیز شامل می شود. . طول مسیر شامل وزن گره ها و یالهاست.
- طول مسیر بحرانی ، یعنی L_{CP} برابر با $T_{flevel}(n_i)$ یا $T_{blevel}(n_i)$ برای گره شروع می- باشد .
- دیرترین زمان ممکن ((As – Late – As – Possible (ALAP)) یعنی $T_{alap}(n_i)$ برابر است با :

$$T_{alap}(n_i) = L_{CP} - T_{blevel}(n_i) = L_{CP} - T_{flevel}(n_i) - w_n(n_i)$$

الگوریتم زیر زمانبندی یک DAG را روی تعداد محدودی از پردازنده ها نشان می دهد.

The Original MCP Algorithm

Step 1. Compute the ALAP time of each node.

Step 2. For each node, create a list which consists of the ALAP times of the node itself and all its descendants in an ascending order, sort these lists in an ascending lexicographical order, and create a node list according to this order.

Step 3. Schedule the first node in the list to the PE that allows the earliest execution, using the insertion approach. Remove the node from the list and repeat Step 3 until the node list is empty.

هنگام زمانبندی گره ها روی پردازنده ها ، ممکن است حفره هایی میان گره هابوجود آید که علتش وجود وابستگی میان گره هاست. در الگوریتم بالا ، گره روی اولین حفرهٔ موجود زمانبندی می گردد که این روش،روش درجی نام دارد. درصورتیکه گره پس از آخرین گره زمانبندی شده و بدون درنظر گرفتن حفره ها زمانبندی گردد ، روش غیردرجی نام خواهد داشت. روش درجی بهتر از روش غیردرجی اجرا می گردد. پیچیدگی روش غیردرجی با داشتن N گره و P پردازنده، برابر O(NP) است ، درحالیکه برای روش

درجـی $O(N^2)$ مـی باشـد. پیچیـدگی گـام اول (O (E ، پیچیـدگی گـام دوم $O(N^2 \log N)$ و پیچیدگی گام سوم با روش درجی $O(N^2)$ می باشد . بنابراین پیچیدگی این الگـوریتم $O(N^2 \log N)$ خواهد بود .

پیاده سازی MCP بصورت کارا تر در این بخش جزئیات یک پیاده سـازی کـارا بـرای MCP را ارائـه مـی دهیم. ابتدا الگوریتم اصلی را کمی تغییر می دهیم تا پیچیدگی آن را کاهش دهیم. درگـام دوم از الگـوریتم اصلی ، تساویها با استفاده از تمام فرزندان شکسته مـی شـوند. تجربـه نـشان داده اسـت کـه بکـار بـردن تمـام فرزندان برای شکستن تساویها ضروری نیست. بجای آن فقط از فرزندان یک سـطح بـرای شکـستن تـساویها استفاده می کنیم که نتایج مشابهی را دربردارد و پیچیدگی گام دوم را به مقـدار (O (E + N Log N کـاهش می دهد. این الگوریتم برای MCP در زیر آمده است.

The Simplified MCP Algorithm

Step 1. Compute the ALAP time of each node.

Step 2. Sort the nodes in the ascending order of ALAP times. Ties are broken by the child that has the smallest ALAP time.

Step 3. Schedule the first node in the list to the PE that allows the earliest execution, using the insertion approach. Remove the node from the list and repeat Step 3 until the node list is empty.

جزئیات پیاده سازی الگوریتم MCP در زیر آمده است. در گـام اول، بـرای محاسـبهٔ زمـان $T_{alap}(n_i)$ ابتدا $T_{flevel}(n_i)$ محاسبه می گردد.مقادیر $T_{flevel}(n_i)$ از پیمایش روبه بالا با شـروع از هـر یـال گـره خروجی بدست می آیند. در همین زمان طول مسیر بحرانی یعنی L_{CP} نیز محاسـبه مـی گرددکـه مـساوی بـا T_{flevel} برای گره شروع است. آنگاه زمان ALAP از فرمول زیر بدست می آید :

$$w_n(n_i) - T_{flevel}(n_i) - L_{CP} = T_{alap}(n_i)$$

درگام دوم ، گره هـا بترتیـب صـعود زمانهـای ALAP مرتـب مـی گردنـد. تـساویها بـر اسـاس یـک فرزنـد بحرانـی کـه کوچکترین زمان ALAP را دارد ، شکسته مـی شـوند. اگـر چنـدین فرزنـد بحرانـی دارای زمـان ALAP یکـسان باشـند ، تساویها بطور تصادفی شکسته می شوند. این کار با مرتب سازی زمانهـای ALAP توسـعه یافتـه انجـام مـی گیـرد کـه در آن زمانهای ALAP گره و فرزند بحرانی به هم پیوند می یابند. آخـرین و سـختترین گـام ، تمـام گـره هـا را بـرای m پردازنـده

زمانبندی می کند تا طول زمانبندی را کمینه نماید و هر گره را به پردازنده ای تخصیص می دهد و زمان شروع گره روی پردازنده را تعیین می نماید .

---

1. Compute ALAP time for each node n_i, initialize $T_{flevel}(n_i)$=0.
   (a) for each node, set the reference count equal to the number of its children, and initialize the ready-list with all nodes that have no child
   (b) while the ready-list is not empty, get a node n_k from the list
   for each of node n_k's parent node n_j
   decrement n_j's reference count, if it becomes zero, append n_j to the ready-list
   let $T_{flevel}(n_j) = \text{MAX}(T_{flevel}(n_j), T_{flevel}(n_k)+w_n(n_k)+ w_l(n_j,n_k))$
   update the critical path length $L_{CP} = \text{MAX}(L_{CP}, T_{flevel}(n_j)+w_n(n_j))$
   (c) for each node n_i, $T_{alap}(n_i) = L_{CP} - T_{flevel}(n_i) - w_n(n_i)$
2. Sort all nodes.
   (a) for node n_i, compute its extended ALAP time $T'_{alap}(n_i)$, initially $T'_{alap}(n_i) = L_{CP}$
   for each of its child node n_j $T'_{alap}(n_i) = MIN(T'_{alap}(n_i), T_{alap}(n_j))$
   $T'_{alap}(n_i) = (L_{CP} + 1) \times T_{alap}(n_i) + T'_{alap}(n_i)$
   (b) sort all nodes to generate a node-list with the node of the least T'_{alap} value at the top
3. Schedule all nodes onto m target PEs, each node n_i has two attributes:
   $PE(n_i) = k$, node n_i is scheduled on the target PE k, and $T_{start}(n_i)$, start time of node n_i
   every PE has a hole-list, initialized by a hole with $T_{holestart} = 0$ and $T_{holeend} = \infty$
   while the node-list is not empty, get a node n_i from the head of the list
   (a) initially set $T'_{ready} = T''_{ready} = 0, T_{start}(n_i) = \infty, PE_l = 0$
   (b) for each of node n_i's parent node n_j
   if $((t_{msg} = T_{start}(n_j)+w_n(n_j)+w_l(n_j,n_i)) > T'_{ready})$ $\{T'_{ready} = t_{msg}, PE_l = PE(n_j)\}$
   (c) for each of node n_i's parent node n_j, $t_{msg} = T_{start}(n_j) + w_n(n_j)$
   if $(PE_l \neq PE(n_j))$ $t_{msg} = t_{msg} + w_l(n_j,n_i)$
   if $(t_{msg} > T''_{ready})$ $T''_{ready} = t_{msg}$
   (d) for each PE k
   if $(k \equiv PE_l)$ then $T_{ready} = T''_{ready}$ else $T_{ready} = T'_{ready}$
   traverse the hole-list of PE k from the beginning, for each hole
   $T_{avail} = MAX(T_{ready}, T_{holestart})$
   if $((T_{holeend} - T_{avail}) \geq w_n(n_i))$ exit from traverse
   if $(T_{avail} < T_{start}(n_i))$ $\{T_{start}(n_i) = T_{avail}, PE(n_i) = k\}$
   (e) update the start and end time of the hole, split it into two if needed

---

برای شرح مختصر آخرین گام ، اصطلاحات زیر تعریف می گردد :

۱- برای هر گره n_i ، زمان ورود پیغام از گره والدین برابر است با :

$$T_{msg}(n_j) = T_{start}(n_j) + w_n(n_j) + w_l(n_j, n_i)$$

که در آن $T_{start}(n_j)$ زمان شروع گره والدین n_j است .

۲- از منظر گره ، زمان آماده شدن یعنی T_{ready} زودترین زمان آماده شدن گره با درنظر گرفتن تمام زمانهای ورود پیغامهای والدین می باشد.

۳- از منظر پردازنده ها ، زمان دردسترس بودن یعنی T_{avail} زودترین زمان در دسترس بودن و آماده بودن پردازنده با در نظر گرفتن تمام گره های تخصیص یافته به پردازنده می باشد .

٤- زودترین زمان شروع یعنی T_{start} کمترین مقدار T_{avail} در میان پردازنده ها می باشد.

گره هایک به یک بترتیب لیست زمانبندی می گردند. برای هر گره، زمان آماده شدن در هر پردازنده محاسبه می گردد. سپس زمان دردسترس بودن یعنی T_{avail} برای هر پردازنده از طریق جستجو در لیست حفره ها بدست می آید. سرانجام زودترین زمان شروع یعنی T_{start} برای گره مشخص می گردد.برای محاسبهٔ زمان آماده شدن گره n_i باید زمان ورود پیغام از والدینش یعنی $T_{msg}(n_j)$ محاسبه گردد. PE_l پردازنده ای است که گره والدی که بیشترین مقدار T_{msg} را دارد ، روی آن قرار می گیرد. تمامی پردازنده ها باستثناء PE_l زمان آماده شدن یکسانی برابر با $MAX(T_{msg}(n_j))$ دارند . گره n_i ممکن است زمان آماده شدن کوچکتری روی PE_l داشته باشد ، چون وزن یال میان گره فرزند و والد صفر است. بنابراین زمان آماده شدن دوبار محاسبه می گردد. یک بار برای PE_l و بار دوم برای پردازنده های دیگر محاسبه زمان محاسبه می شود.

زمان آماده شدن ، زودترین زمان ممکن برای آماده شدن گره برای اجرا همراه با در نظر گرفتن پیغامهای ورودی است.همچنین فضای کافی برای زمانبندی گره باید وجود داشته باشد. برای انجام جستجوی اولین حفرهٔ موجود یک لیست حفره برای هر پردازنده ساخته می شود . هر حفره در این لیست دارای یک زمان شروع $T_{holestart}$ و یک زمان پایان $T_{holeend}$ است .

درصورتیکه رابطهٔ $T_{holeend} - MAX\ (T_{ready}, T_{holestart}) \geq W_n(n_i)$ برقرار باشد،گره می تواند زمانبندی گردد و $T_{start}\ (n_i)$ معین شود .زودترین زمان شروع کمترین مقدار T_{avail} میان تمامی پردازنده هاست و گره برای پردازنده ای زمانبندی می گردد که زودترین زمان شروع را فراهم آورد. پس از آن زمانهای شروع و پایان حفره بروز رسانده می شود . در حالتی که $T_{ready} > T_{holestart}$ باشد ، حفره دو تکه خواهد گردید .

تحلیل پیچیدگی : در گام اول ، زمان ALAP با استفاده از پیمایش هر یال در گراف محاسبه می گردد که پیچیدگی اش برابر $O\ (E)$ است . گام دوم ، گره ها را بترتیب صعود زمانهای ALAP مرتب می کند که پیچیدگی آن برابر $O\ (N \log N)$ می باشد. تساویها فقط توسط اولین فرزندی که کوچکترین زمان ALAP را دارد ، شکسته می شود. هزینهٔ مقایسهٔ هر فرزند گره برابر $O(|\ CHILD(n_i)\ |)$ است و برای N گره ، هزینه برابر $\sum O(|\ CHILD(n_i)\ |) = O(E)$ خواهد بود . بنابراین پیچیدگی گام دوم بصورت $O\ (E + N \log N)$ است .گام سوم درپی یافتن زمان آماده شدن برای هر گره n_i است که هزینه اش برابر $O(|\ PARENT(n_i)\ |)$ است ، چون تمامی والدهای گره n_i بررسی می گردند . لذا برای N گره هزینه بصورت $\sum O(|\ PARENYT\ (n_i)\ |) = O(E)$ خواهد بود . سپس گره n_i بدنبال زودترین زمان T_{avail} می گردد که برای این کار لیستهای حفرهٔ هر پردازنده را جستجو می نماید .

چون تعداد کل حفره ها کمتر از N است ، پیچیدگی زمانبندی یک گره واحد (N) O است و پیچیدگی گام سوم برای زمانبندی N گره بصورت $O(N^2)$ خواهد بود. بنابراین پیچیدگی الگوریتم MCP برابر با $O(E + N\log N + N^2)$ است .

۸. نتیجه گیری

تبدیل اتوماتیک کد ترتیبی به موازی در دو مرحله اصلی امکانپذیر است. در یک مرحله توازی در بین جملات برنامه. در مرحله دیگر توازی در بین تکرارهای حلقه تشخیص داده می شود. جهت تشخیص توازی در اجرای جملات دو نوع وابستگی داده ای و کنترلی در بین جملات را باید مشخص نمود. مسلما" جملات وابسته قابل اجرا به صورت موازی با یکدیگر نیستند. حاصل تشخیص وابستگیها در قالب گراف وظایف نمایش داده می شود. مساله در اینجا دانه بندی و زمانبندی وظایف است.

مقوله زمانبندی در ارتباط با چگونگی تخصیص بهینه وظایف به پردازنده ها مطرح می شود. زمانبندی را می توان به دو صورت ایستا در قبل از اجرا و پویا در حین اجرا انجام داد. مساله زمانبندی از نوع غیر چند جمله ای سخت است. لذا، روشهای مکاشفه ای و غیر قطعی برای زمانبندی گرافهای بزرگ وظایف می تواند بسیار حایز اهمیت باشند.

مساله تشخیص توازی در بین تکرارهای حلقه به تعیین مکانهایی از حافظه که در یک تکرار حلقه مقداردهی و در تکرارهای بعدی استفاده می شوند منجر می گردد. در اینجا مساله تعیین متغیرهایی که بر اساس تکرارهای حلقه مقدراشان تعیین می شود مطرح است.و همچنین مساله دیگر ارجاع به یک خانه آرایه است که در یک تکرار مقدار دهی و در تکرار دیگر مورد استفاده قرار گرفته است. بدست آوردن

اینگونه متغیرها منجر به حل دستگاه معادلات و نامعادلات خطی و غیر خطی می شود. در حالت خطی می توان از معادلات دیوفانتین برای حل دستگاه معادلات اقدام نمود. با بدست آوردن بردارهای پایه می توان مساله تعیین رابطه در بین بردارهای وابستکی بواسطه تعداد زیاد آنها را از میان برداشت. به این ترتیب فضای یکنواخت وابستگیها در بین تکرارهای حلقه ایجاد می شود.

برای افزایش دانه بندی می توان فضای یکنواخت تکرارهای حلقه را کاشی بندی کرد. برای کاشی بندی می توان از الگوریتم وولف استفاده نمود. در روش وولف با تبدیل فضای تکرارهای حلقه به فضایی با قابلیت تعیین سریع کاشیها عمل می شود. مشکل در اینجا بدست آوردن ماتریس تبدیل است. در مورد تعیین کاشیها، مساله تشخیص کاشیهایی است که می توانند مستقلا" با یکدیگر به اجرا در آیند. برای این منظور از تکیکاهای لبه امواج می توان استفاده کرد.

۹. تعیین وابستگی بین تکرارهای حلقه

هدف محاسبه تکرارهایی است که وابستگی داده ای را ایجاد می کنند باید دستگاه معادله های حاصل از مساوی قرار دادن اندیس ها را حل نمود. از بین مقادیر بدست آمده از حل دستگاه معاده های خطی می بایست آن مقادیری را انتخاب نمود که در سرحد تعیین شده برای حلقه قرار دارند. برای نمونه قطعه کد زیر را در نظر بگیرید:

```
For i := 1 To 1000 Do
   For j := 1 To 1000 Do
      For k := 1 To 1000 Do
         S1: d := X [ i + 2 * k + 5, 4 * k - j]
      EndFor
```

S_2: X [i - j, i + j] := …

EndFor

EndFor

در حلقه فوق، بین دو جمله S_1 و S_2 به شرط وجود حداقل دو تکرار (i_1, j_1, k_1) و (i_2, j_2, k_2) با مقادیر اندیس مساوی در S_1 و S_2 می تواند وابستگی داده ای وجود داشته باشد. به این ترتیب با مساوی قرار دادن اندیس آرایه ها برای دو تکرار (i_1, j_1, k_1) و (i_2, j_2, k_2) در S_1 و S_2، شرط وجود وابستگی به صورت زیل می باشد:

$$\mathbf{i_1 + 2k_1 + 5 = i_2 - j_2}$$
$$\mathbf{4k_1 - j_1 = i_2 + j_2}$$

از حل دو معادله فوق مقادیر (i_2, j_2, k_2) بر اساس مقادیر مختلف (i_1, j_1, k_1) حاصل می گردد. البته تنها مقادیری مورد قبول خواهند بود که در حدود داده شده برای حلقه ها؛ یعنی 1 تا 1000 قرار گیرد. مقادیر مورد قبول برای مثال فوق در جدول شکل ۴ نشان داده شده است. در این جدول بردار وابستگی، فاصله بین مقصد و مبدا را نمایش می دهد.

مبدا وابستگی	مقصد وابستگی	بردار وابستگی
(i_1, j_1, k_1)	(i_2, j_2)	$(i_2 - i_1, j_2 - j_1)$
(16, 1, 12)	(46, 1)	(30, 0)
(16, 1, 13)	(49, 2)	(33, 1)
(16, 1, 14)	(52, 3)	(36, 2)
(16, 1, 15)	(54, 4)	(39, 3)
(16, 3, 13)	(48, 1)	(32, -2)
…	…	…

شکل ۴– وابستگی ها در حلقه های ترتیبی تو در تو

در حالت کلی می توان حلقه های تو در تو و وابستگیهایی که از طریق اندیس حلقه ها در بین تکرارهای حلقه ایجاد می شود را به صورت مشخص نمود:

For $i_1 := L_1$ **To** U_1 **Do Step** ST_1

For $i_2 := L_2$ **To** U_2 **Do Step** ST_2

…

For $i_p := L_p$ **To** U_p **Do Step** ST_p

…

For $i_n := L_n$ **to** U_n **Do Step** ST_n

S_1: $X [h_1 (i_1, i_2, \ldots, i_n), h_2 (i_1, i_2, \ldots, i_n), \ldots, h_m (i_1, i_2, \ldots, i_n)] := \ldots$

S_2: $\ldots := X [g_1 (i_1, i_2, \ldots, i_n), g_2 (i_1, i_2, \ldots, i_n), \ldots, g_m (i_1, i_2, \ldots, i_n)]$

EndFor

EndFor

…

EndFor Endfor

برای تعیین وابستگی بین تکرارهای حلقه می بایست اندیسهای آرایه X را برای دو تکرار مختلف با یکدیگر مساوی قرار داد. بدین ترتیب دستگاه معادلات و نامعادلات ذیل حاصل می شود:

$$L_p \le i_p, j_p \le U_p, \quad p = 1, 2, \ldots, n$$

$$h_q (i_1, i_2, \ldots, i_n) = g_q (j_1, j_2, \ldots, j_n), \quad q = 1, 2, \ldots, m$$

در رابطه فوق سرحد هر حلقه ممکن است تابعی از شمارنده یا اندیس حلقه های در برگیرنده آن باشد. به عبارت دیگر می توان سرحد حلقه در عمق P را به صورت زیر مشخص نمود:

$$L_p = L_p (i_1, i_2, \ldots, i_{P-1}),$$

$$U_p = U_p (i_1, i_2, \ldots, i_{P-1}),$$

$$ST_p = ST_p(i_1, i_2, \ldots, i_{P-1}), \quad p = 1, 2, \ldots, n$$

برای ساده تر شدن مساله با انجام یک سری تبدیلات سرحد پایینی شمارنده حلقه را صفر یا ۱ مـی کننـد. همچنین گام یا مقدار اضافه شونده در هر تکرار به شمارنده حلقه را مساوی با ۱ می کننـد. بـه ایـن عمـل در اصطلاح نرمالسازی حلقه گفته می شود. برای نمونه حلقه

For i := m_1 To m_2 Step m_3 do Loop-Body

به حلقه ذیل تبدیل می شود.

For i := 1 To $(m_2 - m_1 + m_3)$ div m_3 Step 1 do $Loop\_Body_{Norm}$

در داخل بدنه حلقه هر جا که شمارنده حلقه i ظاهر می شود آنرا با عبارت ذیل باید جایگزین نمود:

$m_1 + \max((m_2 - m_1 + m_3) \text{ div } m_3, 0) * m_3$

همانگونه که مشاهده می شود از مساوی قرار دادن انـدیس آرایـه هـا تعـدادی معادلـه بدسـت مـی آیـد و همچنین با در نظر گرفتن سر حد آرایه ها تعدادی نا معادله بدست می آید. بنـابراین مـساله تعیـین تکرارهـای وابسته به حل دستگاه معادلات و نامعادلات صحیح منتج شود. روشهایی مثـل حـذف فوریـه-مـوتخین [۱۱، ۱٤] و امگا [٦٤ و ۸٤] فقط نامعادلات حاصل از سرحد حلقه ها را حل مـی کننـد. اینهـا در واقـع محـدوده شماره تکرارهای وابسته را مشخص می کنند. برای بدست آوردن شماره تکرار هـای وابـسته روشـهایی مثـل حل معادلات دیوفاتین[۳٦] خطی [۱۱، ۱٤، ۱٦، ۸٤ و ۹۳]، روش حـذف گوسـی صـحیح[۳۷] [۲۷ و ۸٤] و دلتـا [٤۳] را می توان مورد استفاده قرار داد. همچنین روش های دیگری مثـل متغیـر شـاخص صـفر[۳۸] [٤۳ و ۸٤]، متغیر شاخص منفرد[۳۹] [۸٤]، متغیر شـاخص چنـد تـایی[٤۰] [٤۳ و ۸٤]، [٤۳ و ۸٤]، λ [۸٤]، سـیمپلکس اولیـه و دوتـایی صـحیح [۲۷ و ۵۷] نیـز از معادلـه هـا و هـم از نامعادلـه هـا بـرای تحلیـل وابـستگی داده ای اسـتفاده می کنند.

[36] Diophantine equations
[37] Integer gaussian elimination
[38] Zero index variable (ZIV)
[39] Single index variable (SIV)
[40] Multiply index variable (MIV)

۹-۱ ماتریس اخلون

برای تعیین وابستگی بین تکرارهای حلقه همانطور که در بالا توضیح داده شد نیاز به حل دستگاه معادلات خطی حاصل از متساوی قرار دادن اندیس آرایه ها است. برای این منظور باید ابتدا ماتریس را به فرم Echelon تبدیل نمود. بنا به تعریف یک ماتریس در صورتی در فرم Echelon است که تمام عناصر زیر اولین ستون غیر صفر هر سطر از ماتریس همگی صفر باشند. هر ماتریسی را می توان با یک سری عملیات سطری مناسب مثل ضرب یک سطر در منهای ۱، جابجا کردن سطرها و جمع کردن مضربی از یک سطر با سطر دیگر به ماتریس اخلون تبدیل نمود.

برای محاسبه رنک یا مرتبه، دترمینان و معکوس یک ماتریس می توان آن را به فرم اخلون تبدیل نمود.

ماتریس اخلون دارای دو ویژگی ذیل است:

۱- سطهای صفر در پایین قرار دارند

۲- سمت چپ ترین عنصر غیر صفر در هر سطر در سمت راست عنصر غیر صفر سطح پایین آن قرار دارد.

برای تبدیل یک ماتریس به فرم اخلون معمولاً سه عمل ذیل رایج است:

۱- تعویض دو سطر

۲- ضرب یک مقادیر یک سطر در عددی مخالف صفر

۳- جایگزینی یک سطر با حاصل جمع آن سطر با سطرهای دیگر

با استفاده از سه عمل ساده فوق می توان ماتریسها را تبدیل به فرم اخلون نمود. در ذیل مراحل تبدیل به فرم اخلون ارایه شده است. روش تبدیل را حذف گوسی می نامند.

۱- سطوح ماتریس جابجا شود تا سطرها با سمت چپ ترین عنصر غیر صفر در بالای ماتریس قرار بگیرند.

۲- مضربی از هر سطر را با مضربی از بالاترین سطر با سمت چپ ترین عنصر غیر صفر جمع نمایید تا عنصری از صطر که زیر عنصر غیر صفر قرار می گیرد مساوی با صفر شود.

۳- تکرار مراحل فوق تا تبدیل ماتریس به فرم اخلون

برای تبدیل ماتریس A به ماتریس S در فرم Echelon ، ماتریس تبدیل U ایجاد می شود به قسمی که $S_{m\times n} = U_{m\times m}A_{m\times n}$ می شود. برای این منظور ابتدا ماتریس U به صورت یک ماتریس واحد یا unity که عناصر قطر اصلی آن یک و سایر عناصر صفر هستند مشخص می شود. هر نوع عملیات سطری که بر روی A جهت تبدیل آن به فرم اخلون انجام می گیرد نیز بر روی U باید انجام شود. برای نمونه به مثال ذیل توجه نمایید:

$$A \approx \begin{bmatrix} 2 & 3 \\ 7 & 1 \\ 3 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 \\ 7-6 & 1-4 \\ 3 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 \\ 1 & -3 \\ 3 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 \\ 3 & 2 \\ 1 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 \\ 3-3 & 2+9 \\ 1 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 \\ 1 & -3 \\ 0 & 11 \end{bmatrix}$$

$$U = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \Rightarrow \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{matrix} \Rightarrow \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{matrix} \Rightarrow \begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -2 \end{matrix} \Rightarrow \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & -5 \\ 0 & 1 & -2 \end{matrix} \Rightarrow \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 1 & -5 \end{matrix}$$

(1) $row_2 = row_2 - 2*row_3$, (2) interchange row_2, row_3 ,

(3) $row_2 = row_2 + 3*row_3$, (4) interchange row_2, row_3

الگوریتم تبدیل ماتریس A با ابعاد m× n به صورت زیر است.

Set $U \leftarrow I_m$, $S \leftarrow A$, $i_0 \leftarrow 0$
for j := 1 to n do
  begin
    If there is at least one non zero S_i with $i0 \le i \le m$
    then begin
        set $i_0 \leftarrow i_0 + 1$
        for i := m to n step -1 do
            while $S_{i,j} \neq 0$ do
              begin
                set $\alpha \leftarrow sig(S_{i-1,j} * S_{i,j})$
                    $\beta \leftarrow \lfloor |S_{i-1,j}| / |S_{i,j}| \rfloor$
                 Subtract $\alpha * \beta *$ row_i from row_{i-1} in both the U and S matrices
                 Interchange row_i and row_{i-1} in both the U and S matrices
              end
        end\
  end

مثال :

$$A = \begin{matrix} 4 & 4 & 1 \\ 6 & 0 & 1 \\ 4 & 3 & 2 \end{matrix} \qquad U = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \qquad S = A = \begin{matrix} 4 & 4 & 1 \\ 6 & 0 & 1 \\ 4 & 3 & 2 \end{matrix}$$

(1) $j = 1, i0 = 1, i = 3, \alpha = +1, \beta = \lfloor 6/4 \rfloor = 1$

در تکرار اول حلقه فوق از الگوریتم فوق سطر سوم از دوم کم می شود:

$$S = \begin{matrix} 4 & 4 & 1 \\ 6 & 0 & 1 \\ 4 & 3 & 2 \end{matrix} \qquad U = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{matrix}$$

دو سطر سوم و دوم با یکدیگر جابجا می شوند:

$$S = \begin{matrix} 4 & 4 & 1 \\ 4 & 3 & 2 \\ 6 & 0 & 1 \end{matrix} \qquad U = \begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -1 \end{matrix}$$

(2) $j = 1, i0 = 1, i = 3, \alpha = +1, \beta = \lfloor 4/2 \rfloor = 2$

در تکرار دوم حلقه در الگوریتم فوق دو برابرسطر سوم از دوم کم می شود:

$$S = \begin{matrix} 4 & 4 & 1 \\ 0 & 9 & 4 \\ 2 & -3 & -1 \end{matrix} \qquad U = \begin{matrix} 1 & 0 & 0 \\ 0 & -2 & 3 \\ 0 & 1 & -1 \end{matrix}$$

دو سطر سوم و دوم با یکدیگر جابجا می شوند:

$$S = \begin{matrix} 4 & 4 & 1 \\ 2 & -3 & -1 \\ 0 & 9 & 4 \end{matrix} \qquad U = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -2 & 3 \end{matrix}$$

(3) $j = 1, i_0 = 1, i = 2, \alpha = +1, \beta = \lfloor 4/2 \rfloor = 2$

در تکرار سوم حلقه در الگوریتم فوق دو برابرسطر سوم از سطر اول کم می شود:

$$S' = \begin{matrix} 0 & 10 & 3 \\ 0 & 9 & 4 \\ 2 & -3 & -1 \end{matrix} \qquad U = \begin{matrix} 1 & -2 & 2 \\ 0 & 1 & -1 \\ 0 & -2 & 3 \end{matrix}$$

دو سطر دوم و اول با یکدیگر جابجا می شوند:

$$S' = \begin{matrix} 2 & -3 & -1 \\ 0 & 10 & 3 \\ 0 & 9 & 4 \end{matrix} \qquad U = \begin{matrix} 0 & 1 & -1 \\ 1 & -2 & 2 \\ 0 & -2 & 3 \end{matrix}$$

(4) $j = 2, i_0 = 2, i = 3, \alpha = +1, \beta = \lfloor 10/9 \rfloor = 1$

در تکرار چهارم حلقه در الگوریتم فوق سطر سوم از سطر دو کم می شود:

$$S' = \begin{matrix} 2 & -3 & -1 \\ 0 & 10 & 3 \\ 0 & 9 & 4 \end{matrix} \qquad U = \begin{matrix} 0 & 1 & -1 \\ 1 & -2 & 2 \\ 0 & -2 & 3 \end{matrix}$$

اکنون سطر سوم از دوم کم می شود:

$$S' = \begin{matrix} 2 & -3 & -1 \\ 0 & 1 & -1 \\ 0 & 9 & 4 \end{matrix} \qquad U = \begin{matrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & -2 & 3 \end{matrix}$$

اکنون دو سطر سوم و دوم جابجا می شوند:

$$S' = \begin{matrix} 2 & -3 & -1 \\ 0 & 9 & 4 \\ 0 & 1 & -1 \end{matrix} \qquad U = \begin{matrix} 0 & 1 & -1 \\ 0 & -2 & 3 \\ 1 & 0 & -1 \end{matrix}$$

(5) $j = 2, i_0 = 2, i = 3, \alpha = +1, \beta = \lfloor 9/1 \rfloor = 9$

در تکرار پنجم حلقه در الگوریتم فوق، ۹ برابر سطر سوم از سطر دو کم می شود:

$$S' = \begin{matrix} 2 & -3 & -1 \\ 0 & 0 & 13 \\ 0 & 1 & -1 \end{matrix} \qquad U = \begin{matrix} 0 & 1 & -1 \\ -9 & -2 & 12 \\ 1 & 0 & 1 \end{matrix}$$

اکنون سطر سوم و دوم جابجا می شوند:

$$S' = \begin{matrix} 2 & -3 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 13 \end{matrix} \qquad U = \begin{matrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -9 & -2 & 12 \end{matrix}$$

۹-۲ حل معادلات دیوفانتین

معادلات دیوفانتین معادلاتی هستند با ضرایب صحیح که جوابهای صحیح آنها مورد نظر می باشد. معادله دیوفانتین خطی در حالت کلی به صورت ذیل می باشد:

$$a_1 x_1 + a_2 x_2 + \ldots + a_m x_m = c \qquad (1)$$

در معادله فوق m ضریب صحیح a_1 تا a_m و m مجهول x_1 تا x_m که نیز اعداد صحیح هستند، مطرح است. مقادیر حاصل از این معادله اعداد صحیح می باشند، اگر و تنها فقط اگر c ضریبی از بزرگترین مضرب مشترک ضرایب a_1 تا a_n باشد. به عبارت دیگر:

$$Gcd(a_1, a_2, \ldots, a_n) \text{ divides } c$$

هدف حل دستگاه معادلات ذیل است:

$$\begin{aligned}
&a_{11} x_1 + a_{21} x_2 + \ldots + a_{m1} x_{m1} = c_1 \\
&a_{12} x_1 + a_{22} x_2 + \ldots + a_{m2} x_{m1} = c_2 \\
&\quad \ldots \qquad\qquad\qquad\quad \ldots \\
&\quad \ldots \qquad\qquad\qquad\quad \ldots \\
&\quad \ldots \qquad\qquad\qquad\quad \ldots \\
&a_{1n} x_1 + a_{2n} x_2 + \ldots + a_m x_{mn} = c_n
\end{aligned}$$

دستگاه معادلات فوق را می توان در فرم ماتریسی به صورت ذیل مشخص نمود:

$$XA = C \quad \text{Where } X = (x_1, x_2, \ldots, x_m)$$
$$C = (c_1, c_2, \ldots, c_m)$$

$$A = \begin{matrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \ldots & & & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{matrix}$$

برای حل ماتریس A به فرم اخلون S می بایست ماتریس تبدیل U را محاسبه نمود. در این صورت:

$$S_{m\times n} = U_{m\times m} A_{m\times n}$$

با تبدیل ماتریس به فرم اخلون در واقع آرایه $t_{1\times m}$ به عنوان جواب اپر حاصل شود آنگاه:

$$t_{1\times m} * S_{m\times n} = C_{1\times n} \Rightarrow t_{1\times m} * U_{m\times m} A_{m\times n} = C_{1\times n}$$

از طرف دیگر :

$A_{m\times n}X_{m\times 1} = C_{1\times n} \Rightarrow X_{m\times 1} = t_{1\times m}\, U_{m\times m}$

برای نمونه به مثال ذیل توجه نمایید.

```
For I := 1 to 1000 do
    For j := 1 to 1000 do
      begin
        S1: A[i+j, 3*i+j+3] = …
        S2: … = … A[i+j+1, i+2*j+4];
      end
```

برای بدست آوردن تکرارهای وابسته ابتدا دستگاه معادلات تشکیل می شود.

$I1 + j1 = i2 + j2 + 1 \Rightarrow I1 + j1 - i2 - j2 - 1 = 1$

$3*I1 + j1 + 3 = I2 + 2*j2 + 4 \quad 3*I1 + j1 + 3 - I2 - 2*j2 = 1$

$$A = \begin{matrix} 1 & 3 \\ 1 & 1 \\ -1 & -1 \\ -1 & -2 \end{matrix}, \qquad C = (1, 1), \quad X = (I1, j1, I2, j2) \ , \ XA = C$$

اکنون ماتریس اخلون طبق الگوریتم ارایه شده در بالا ایجاد می شود و بر اساس ماتریس S ماتریس U بصورت ذیل ایجاد می شود:

$$U = \begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 0 \end{matrix} \qquad S = \begin{matrix} 2 & 1 \\ 0 & 3 \\ 0 & 0 \\ 0 & 0 \end{matrix}$$

$(t1, t2, t3, t4)*S = (1,1) \Rightarrow t1 = -1, t2 = -1, t3,t4$

$$X = t*U = (-1,-1,t3,t4)*\begin{vmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 0 \end{vmatrix} = (t3,\ t4,\ -t3 + t4 - 1,\ 2t3)$$

$X = (I1, j1, I2, j2) \Rightarrow I1 = t3,\ j1 = t4,\ I2 = t4 - t3 - 1,\ j2 = 2t3$

اکنون با تعیین مقادیر برای t3 و t4 با در نظر گرفتن سرحد مقادیر I1 و j1 درون حلقه می توان تکرارهای وابسته را مشخص نمود.

همانگونه که در بالا توضیح داده شد، آزمونهای وابستگی معمولاً منجر به حل دستگاه معادلات و نامعادلات خطی می شوند. لذا، از معادلات دیوفانتین برای بدست آوردن پاسخ برای معادلات و با استفاده از حذف فوریه برای حل نامعادلات استفاده خواهد شد. معادله دیوفانتین خطی در حالت کلی به صورت ذیل می باشد:

$$a_1 x_1 + a_2 x_2 + ...+ a_n x_n = c \qquad (1)$$

مقادیر حاصل از این معادله اعداد صحیح می باشند، اگر و تنها فقط اگر C ضریبی از بزرگترین مضرب مشترک ضرایب a_1 تا a_n C باشد. به عبارت دیگر:

$$Gcd(a_1, a_2, ..., a_n) \text{ divides } c$$

می توان با استفاده از استقرا این مساله را اثبات نمود. بدین ترتیب که برای حالتی که تعداد متغیرها یک عدد است معادله به صورت:

$$a_1 x_1 = c$$

می باشد. این معادله دارای جواب صحیح می باشد اگر و تنها فقط اگر C بر a_1 قابل تقسیم باشد یا در اصطلاح آنرا عاد نماید. اگر کوچکترین ضریب برای مثال a1=1 باشد، آنگاه $gcd(a_1, a_2, ..., a_n)$ مساوی با ۱ خواهد بود. در این صورت رابطه (۱) دارای جواب ذیل است:

$$(c- a_2 t_2 – a_3 t_3 - ....+ a_n t_n , t_2, t_3, \ldots, t_n)$$

در حالت کلی فرض کنید که a_1 کوچکترین اندیس باشد و داشته باشیم:

$$t = x_1 + \lfloor a_2 / a_1 \rfloor x_2 + ...+ \lfloor a_n / a_1 \rfloor x_n \qquad (2)$$

می توان رابطه (۱) را به صورت ذیل بازنویسی نمود:

$$a_1 x_1 + a_2 x_2 + ...+ a_n x_n = c$$
$$a_1 x_1 + a_1 * \lfloor a_2 / a_1 \rfloor + (a_2 \bmod a_1) x_2 + ...+ a_1 * \lfloor a_n / a_1 \rfloor + (a_n \bmod a_1) x_n = c$$

با جایگزینی t در رابطه فوق رابطه (۳) به صورت ذیل بدست می آید:

$$a_1 t + (a_2 \bmod a_1) x_2 + ...+ (a_n \bmod a_1) x_n = c \qquad (3)$$

رابطه فوق دارای جواب صحیح است اگر و تنها فقط اگر رابطه (۱) دارای جواب صحیح باشد. اکنون مشخص است که :

$gcd(a,b) = gcd(a \bmod b, b) \Rightarrow gcd(a_1,a_2,...,a_n) = gcd(a_1, (a_2 \bmod a_1),..,(a_n \bmod a_1))$
$\Rightarrow gcd(a_1, (a_2 \bmod a_1),..,(a_n \bmod a_1))$ divides c.

به این ترتیب a_1 چنانچه در رابطه (۱) کوچکترین ضریب باشد، بقیه ضرایب می بایست صفر باشند زیرا بقیه ضرایب باقیمانده تقسیم بر a_1 را مشخص می کنند که مسلماً از a_1 کوچکتر است. در غیر اینصورت مقدار کوچکترین ضریب کاهش بافته که نشانگر موفقیت در استقرا است. به مثالهای ذیل توجه نمایید:

(1) $2x = 3$ => هیچ جوابی ندارد

(2) $2x = 6$ => $x = 3$

(3) $2x + y = 3$ => $x = t, y = 3 - 2t$ Gcd(2, 1) = 1 which divides 3

(4) $2x + 3y = 3$ => $z = t, y = 3 - 2t$ => $x = 3t - 3, \quad y = 3 - 2t$

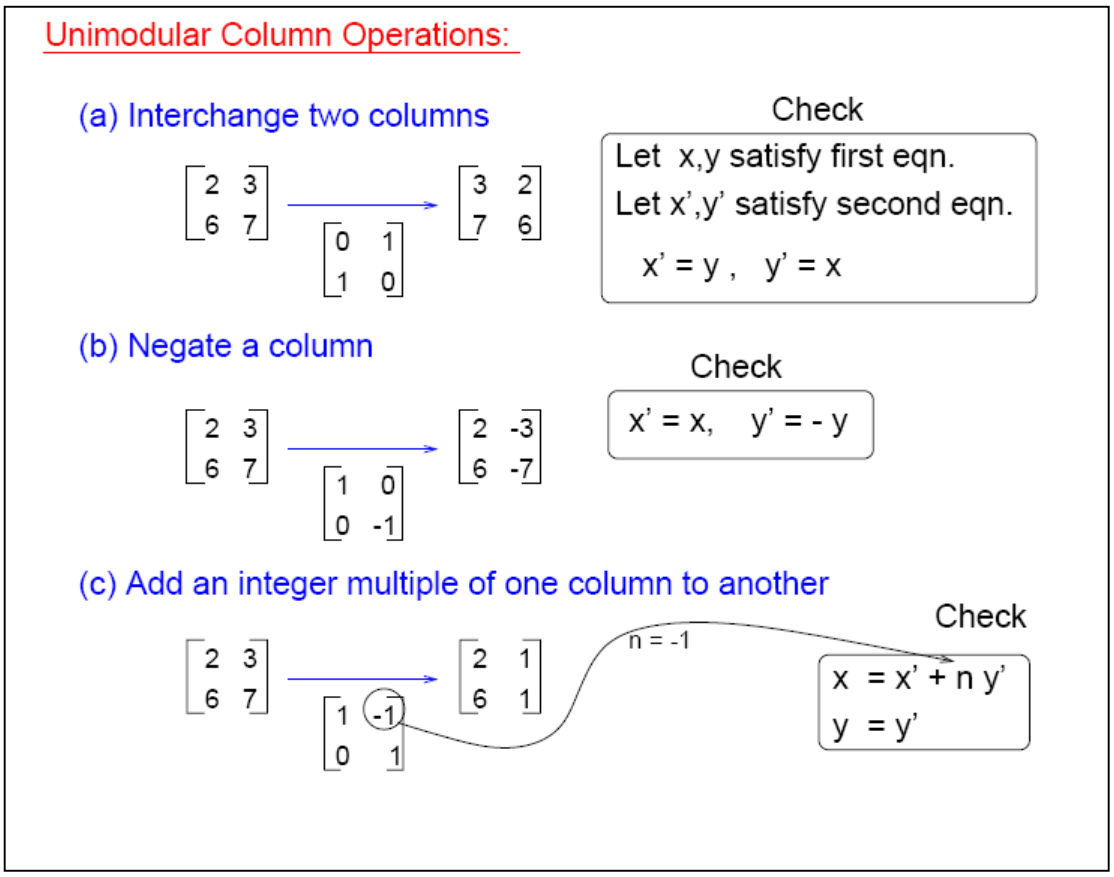
برای حل دستگاه معادلات خطی $Ax = b$ می توان $U1, U2, \ldots, Uk$ را به قسمی مشخص نمود که رابطه ذیل برقرار گردد:

$Ax = b$
$L = A*U1*U2*...*Uk$ is lower triangular
Solve $Lx' = b$ (easy)
Compute $x = (U1*U2*...*Uk)*x'$

اثبات:

$(A*U1*U2...*Uk)x' = b \Rightarrow A(U1*U2*...*Uk)x' = b \Rightarrow x = (U1*U2...*Uk)x'$

برای نمونه به مثال ذیل توجه نمایید:



شکل ۵- نمونه ای از حل معادلات دیوفانتین با استفاده از حذف گوسی

به عنوان نمونه ای دیگر به مثال ذیل توجه نمایید:

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{matrix} 5 \\ 5 \end{matrix}$$

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & -1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 & 0 \\ 1 & -1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 1 & 0 \\ 1 & -2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 5 & -2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ -2 & 5 & 0 \end{bmatrix}$$

$$\begin{vmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 5 & 0 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \Rightarrow \begin{matrix} x' = 5 \\ y' = 3 \\ z' = t \end{matrix} \Rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1 & 3 & -2 \\ 1 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ t \end{bmatrix} = \begin{bmatrix} 4-2t \\ -1 \\ t \end{bmatrix}$$

شکل ۶- نمونه ای از حل معادلات دیوفانتین با استفادع از ماتریس یونیماجولار

تعویض ستونها، منفی کردن یک سطر و بالاخره جمع و تفریق مضربی از سطرها با یکدیگر راه مثلثی نمودن ماتریس ضرایب است.

مقادیر حاصل از این معادله اعداد صحیح می باشند، اگر و تنها فقط اگر c ضریبی از بزرگترین مضرب مشترک ضرایب a_1 تا a_n باشد. به عبارت دیگر برای حل دستگاه نامعادله ها و اعمال محدودیتها که در واقع سرحدهای حلقه ها ایجاد می کنند، از روش حذف فوریه استفاده می شود که در ادامه ارائه خواهد شد.

۴-۶-۲ حل دستگاه معادلات و نامعادلات

در این قسمت روشهایی برای حل دستگاه شامل چند نامعادله و معادله ارائه می شود. روشهای ارائه شده تحت شرایط خاص عمل نموده، کلی نیستند. چنانچه دو جمله S1 و S2 به خانه های آرایه X از طریق به ترتیب تابع خطی $h(i_1, i_2, \ldots, i_k)$ و $g(i_1, i_2, \ldots, i_k)$ دسترسی داشته باشند، آنگاه در بین این دو جمله وابستگی داده ای در صورتی وجود دارد که دو تکرار از حلقه وجود داشته باشد به قسمی که:

$$h(i_1, i_2, \ldots, i_k) = g(j_1, j_2, \ldots, j_k)$$

باید توجه داشته باشید که معادله $a_1i_1 + a_2i_2 + \ldots a_ki_k = b$ که در آن ضرایب a_1 اعداد صحیح هستند، در صورتی قابل حل است که b ضریبی از بزرگترین مضرب مشترک ضرایب a_i و یا $gcd(a_1 , a_2 , \ldots a_k)$ باشد.چنانچه رابطه برقرار باشد کل سیستم مستقل خطی است. اثبات موضوع به این صورت است که:

فرض: $a = gcd(a_1 , a_2 , \ldots a_k)$

معادله اولیه: $a_1i_1 + a_2i_2 + \ldots a_ki_k = b$

$a*(a_1/a* i_1 + a_2 /a * i_2 + \ldots + a_n / a* i_n) = b$

در صورتی که معادله قابل حل باشد مقدار داخل پرانتز یک عدد صحیح مثل T خواهد بود. بنابراین:

$a*T = b$

بنابراین می بایست b بر a بخش پذیر باشد. همچنین با در نظر گرفتن اینکه اندیسهای حلقه در بین سرحدهای حلقه قرار می گیرند در نتیجه چنانچه سرحد پایینی با حرف L و سرحد بالایی با U مشخص شده باشد، آنگاه باید :

$$a_1L_1 + a_2L_2 + \ldots a_kL_k \le b \le a_1U_1 + a_2U_2 + \ldots a_kU_k$$

برای استفاده از روش حذف گوس صحیح جهت حل کردن دستگاه نا معادله ها می بایست یکی از ضرایب در معادله ها +۱ یا -۱ باشد. به این ترتیب با حذف گوسی آن متغیر مربوط به آن ضریب می توان دستگاه معادلات را کوچکتر نمود. برای نمونه به مثال ذیل توجه کنید:

$$\begin{aligned} x_1 + x_2 &\le 10 \\ x_2 - 4x_3 &\le 12 \\ -6x_1 + x_2 - 7x_3 &= -4 \\ -4x_1+3x_2 - 2x_3 &= -1 \end{aligned}$$

بر اساس سومین سطر می توان مقدار x_2 را به صورت ذیل محاسبه نمود:

$$x_2 = 6x_1 + - 7x_3 - 4$$

بنابراین با جایگزینی x_2 خواهیم داشت

$$\begin{aligned} 7x_1 + 7x_3 &\le 14 \\ 6x_1 + 3x_3 &\le 16 \\ 14x_1+ 19x_3 &= 11 \end{aligned}$$

بنابراین برای بدست آوردن جواب و استفاده از روش حذف گوسی می بایست روشی جهت تبدیل ضرایب به یک وجود داشته باشد. شاید در هنگام بدست آوردن کوچکترین مضرب مشترک بتوان به این نتیجه رسید که با تقسیم طرفین به ضریب مشترک یکی از متغیرها ضریبش ۱ خواهد شد. روش دیگر برای

تبدیل ضرایب به ۱ ± جمع و تفریق معادله ها و نامعادله های داخل دستگاه شاید موثر واقع شود. برای این منظور چنانچه در حالت کلی بتوان دستگاه معادله های:

$$a_{i1}x_1 + a_{i2}x_2 + \dots a_{ik}x_k = di$$
$$a_{j1}x_1 + a_{j2}x_2 + \dots a_{jk}x_k = dj$$

را با معادله های :

$$\lambda i(a_{i1}x_1 + a_{i2}x_2 + \dots a_{ik}x_k) + \lambda j(a_{j1}x_1 + a_{j2}x_2 + \dots a_{jk}x_k) = \lambda i\ di + \lambda j\ dj$$
$$(\lambda i\ a_{i1} + \lambda j\ a_{j1})\ x_1 + (\lambda i\ a_{i2} + \lambda j\ a_{j2})x_2 + \dots (\lambda i\ a_{ik} + \lambda j\ a_{jk})x_k = \lambda i\ di + \lambda j\ dj$$

به شرط اینکه $\lambda i\ a_{ik} + \lambda j\ a_{jk} = \pm 1$ گردد، جایگزین نمود. تنها هنگامیکه a_{ik} و a_{jk} نسبت به یکدیگر اول باشند می توان به این نتیجه رسید زیرا، در غیر این صورت می توان از بزرگترین مقسوم علیه مشترک آنها فاکتور گرفت. اگر f باشد سمت راست نمی تواند ۱ شود. برای نمونه سیستم ذیل را در نظر بگیرید:

$$x_1 + x_2 \leq 10$$
$$x_2 - 4x_3 \leq 12$$
$$-2x_1 + 5x_2 + 3x_3 = 2$$
$$-4x_1 + 3x_2 - 2x_3 = -1$$

بنابراین با ضرب کردن طرفین اولین معادله با $\lambda 1 = -1$ و ضرب طرفین دومین معادله با $\lambda 2 = 2$ و جمع کردن این دو معادله می توان به این نتیجه رسید که :

$$-6x_1 + x_2 - 7x_3 = -4$$

بنابراین دستگاه به صورت ذیل تبدیل می شود.

$$x_1 + x_2 \leq 10$$
$$x_2 - 4x_3 \leq 12$$
$$-6x_1 + x_2 - 7x_3 = -4$$
$$-4x_1 + 3x_2 - 2x_3 = -1$$

اکنون می توان مقدار x_2 را از طریق معادله ها محاسبه و جایگزین نمود.

در روش فوق نیاز بود که ضرایب نسبت به یکدیگر اول باشند. برای این منظور دستگاه معادله ها را به صورت زیر می توان تبدیل نمود.

$$a_1x_1 + a_2x_2 + \dots\ a_kx_k = d_1 \quad , t_1 = a_1x_1 + a_2x_2 \ \Rightarrow\ t_1 + \dots\ a_kx_k = d_1$$

با فرض اول بودن a1 و a2 نسبت به یکدیگر می توان λ1 و λ2 را بگونه ای مشخص نمود که رابطه ذیل برقرار گردد:

$$\lambda_1 a_1 + \lambda_2 a_2 = 1$$

با در نظر گرفتن :

$$t_2 = b_1x_1 + b_2x_2$$

where $b_1 = -\ \lambda_2$, $b_2 = \lambda_1$

اکنون با در نظر گرفتن دو رابطه ذیل :

$$t_1 = a_1x_1 + a_2x_2$$
$$t_2 = b_1x_1 + b_2x_2$$

برای بدست آوردن مقادیر x1 و x2 بر اساس t1 و t2 لازم است که $a_1b_2 - a_2b_1 = \pm 1$ باشد. علت را می توان در حل

$$\Rightarrow \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{\det(A)} \times A^* \times \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

$$\Rightarrow \text{ If } \det(A) = a_1b_2 - a_2b_1 = \pm 1 \quad \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_2 & -a_2 \\ -b_1 & a_1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

$$\Rightarrow$$

$$t_1 = a_1\, x_1 + a_2\, x_2$$
$$t_2 = b_1\, x_1 + b_2\, x_2$$

و خواهیم داشت :

$$x_1 = \quad b_2\, t_1 - a_2\, t_2$$

$$x_2 = -b_1 t_1 + a_1 t_2$$

بنابراین می توان x_1 و x_2 را با t_1 و t_2 جایگزین نمود. به این ترتیب خواهیم داشت:

$$t_1 = d_1 - a_3x_3 \ldots - a_kx_k$$

باید توجه نمود که از رابطه $a_1b_2 - a_2b_1 = \pm 1$ می توان به این نتیجه رسید که محدودیت این روش در این است که باید بزرگترین مقسوم علیه مشترک a_1 و a_2 مساوی با ۱ باشد. برای نمونه به مثال قبلی توجه نمایید.

$$x_1 + x_2 \leq 10$$
$$x_2 - 4x_3 \leq 12$$
$$-2x_1 + 5x_2 + 3x_3 = 2$$
$$-4x_1 + 3x_2 - 2x_3 = -1$$

با فرض : $t1 = -4x_1+3x_2 \Rightarrow a_1=-4 \ , \ a_2=3$

با توجه به $\lambda_1 a_1 + \lambda_2 a_2 = 1$ داریم $\lambda_1 = -1 \ , \ \lambda_2=-1$ و بنابراین $b1=1$ و $b2=-1$ و در نتیجه $t_2=x_1-x_2$ و در نهایت داریم :

$$x_1 = -t_1 - 3 t_2$$
$$x_2 = -t_1 - 4 t_2$$

حال با جایگزینی x_1 , x_2 در معادلات فوق ۴ معادله خواهیم داشت که در یکی از آنها ضریب متغیر t_1 برابر یک خواهد بود که می توان همانند قبل اقدام به حذف آن نمود.

$$-2 t_1 - 7t_2 \leq 10$$
$$- t_1 - 4t_2 - 4 x_3 \leq 12$$
$$-3 t_1 - 14t_2 + 3 x_3 = 2$$
$$t_1 - 2 x_3 = -1$$

اکنون می توان حذف گوسی را بر t_1 اعمال نمود.

چنانچه بزرگترین مقسوم علیه مشترک a_1 و a_2 مساوی با p باشد، آنگاه می توان ضریب t_1 را به قسمی محاسبه نمود که :

$$pt_1 = a_1x_1 + a_2x_2 \Rightarrow t_1 = (a_1/p)x_1 + (a_2/p)x_2$$

که در این نتیجه به مورد قبلی می رسیم. اکنون λ_1 و λ_2 را آنچنان محاسبه می کنیم که :

$$\lambda_1 (a_1/t_1) + \lambda_2 (a_2/t_2) = 1$$

شود. حالا می توان تغییر متغیر ذیل را اعمال نمود.

$$t_1 = (a_1/p)x_1 + (a_2/p)x_2$$
$$t_2 = \lambda_1 x_1 + \lambda_2 x_2$$

بنابراین

$$\begin{cases} x_1 = \lambda_1 t_1 - (a_1/p)t_1 \\ x_2 = \lambda_2 t_2 - (a_2/p)t_2 \end{cases}$$

۳-۶-۴ حل دستگاه نامعادلات با روش حذف فوریه

در این بخش امکان ساده سازی دستگاه نامعادله ها مورد بررسی قرار می گیرد. روش حذف فوریه برای جوابهای صحیح بطور دقیق پاسخگو نیست. برای بکارگیری این روش ابتدا باید x1 از سایرین مجزا شود.به قسمیکه سرحدهای پایینی و بالایی برای x1 بر اساس سایر متغیرها مشخص شود. مسلم است که کلیه سرحدهای پایینی برای یک متغیر باید از کلیه سرخد بالایی کمتر باشنند. در غیر اینصورت یک تناقض وجود داد. از روش حذف فوریه در این قسمت برای کاهش سیستم با حذف متغیرها عمل می شود. برای این منظور به صورت زیر اقدام می شود:

۱- در هر نامعادله x1 جدا می شود به قسمیکه برای آن سرحدهای پایینی و بالایی بر اساس سایرین مشخص می شود.

۲- سیستم در صورتی سازگار خواهد بود که سرحدهای پایینی کوچکتر از بالایی باشند.

۳- به این ترتیب از درون نامعادله ها x1 حذف می شود . همین عمل برای سایرین تکرار می شود.

٤- چنانچه در ضمن حذف تناقضی مشاهده نشود پاسخ وجود دارد که ممکن است اعشاری باشد.

فرض کنید تعداد نامعادله ها m باشد این نامعادله ها را به سه دسته تقسیم می توان نمود. دسته اول شامل متغیرها با ضریب مثبت برای x1 به تعداد m1 عدد است. دسته دوم شامل نامعادله ها با ضریب منفی برای x1 به تعداد m2 عدد است. دسته سوم شامل نامعادله هایی است که اصلاً x1 در آنها وجود ندارد. به این ترتیب تعداد m2 حد پایینی و m1 حد بالایی برای x1 می توان مشخص نمود. سیستمی شامل m1*m2 نامعادله ایجاد می شود. زیرا هر حد پایینی باید از کلیه سرحدهای بالایی کوچکتر باشد و بالعکس. پس از جایگزینی x1 در صورتی سیستم نامعادلات مشابه قبلی خواهد بود که :

معیار ۱- مقدار هر ضریب مثبت از x1 مساوی ۱ باشد

معیار ۲- مقدار هر ضریب منفی از x1 مساوی -۱ باشد

بنابراین در صورتی می توان عمل جایگزینی را انجام داد که یکی از دو معیار فوق برقرار باشد. تعداد نامعادله ها در اینجا هنوز به عنوان یک مشکل مطرح است.واضح است که تعداد نامعادله ها در موارد زیر اضافه نمی شود:

مورد ۱- $m1 \le 1$

مورد 2- $m2 \le 1$

مورد 3- $m1 = 2 \wedge m2 = 2$

۴-۶-۴ معرفی متغیرهای غیر منفی

در حالت کلی مسائل برنامه ریزی خطی نیازمند این هستند که مقدار متغیرها مثبت باشد. بنا براین برای حل دستگاه نامعادله ها آنها را تبدیل به دستگاهی باید نمود که متغیرها همه در آن مثبت هستند. با افزایش مقادیر مثبت به سیستم نامعادله ها می توان آنها را تبدیل به معادله نمود. و با استفاده از روش حذف گوسی می توان از داخل نامعادله ها متغیرهای قبلی را خذف نمود و دستگاه را تبدیل به تعدادی نامعادله با متغیرهای مثبت نمود. فرض کنید سیستم از نامعادله ها به صورت ذیل موجود است:

۴-۶-۴ حل نهایی دستگاه نامعادله ها و معادله ها

با استفاده از تکنیکهایی که مبتنی بر الگوریتم شناخته شده سیمپلکس هستند در این قسمت مبادرت به حل دستگاهی از متغیرها با

$$\sum_{j=1}^{n} c_j x_j = z$$

مقادیر مثبت می شود. هدف در روش سیمپلکس حل مساله بهینه سازی ذیل است:

$$\sum_{j=1}^{n} a_{i,j} x_j \le d_i$$

$$\min z = 2^0$$

$$x_j \ge 0 \qquad j \in \{1,2, ..., n\}$$

$i \in \{1,2$ باشد. الگوریتم دوجنبه ای سیمپلکس d_i ها غیر منفی d_i الگوریتم سیمپلکس را هنگامی می توان مورد استفاده قرار داد که مقادیر را زمانی می توان بکار گرفت که ضرایب c_j غیر منفی هستند. برای ارجاع به هر نامعادله یک متغیر ظاهری ایجاد می شود. به این تریب در اولین مرحله دستگاه به صورت زیر مشخص می شود.

$$Z = c_1x_1 + c_2x_2 + \ldots - 0$$

$$x_{n+i} : a_{i1} x_1 + a_{i2} x_2 + \ldots \le d_i$$

متغیرهایی که درون تابع Z ظاهر می شوند به عنوان متغیرهای خارج از زمینه و متغیرهایی که درون نامعادله ها مشخص می شوند به نام متغیرهای زمینه مشخص می شوند. الگوریتم سیمپلکس مبتنی بر این مشاهده است که اگر ضرایب c_j غیر منفی باشند . همچنین ضرایب d_i غیر منفی باشند آنگاه مساله با مقدار صفر به هر متغیر خارج از زمینه و مقدار d_i برای هر متغیر زمینه حل خواهد شد. با تغییر متغیرها می توان این دو شرط را برای سیستم ایجاد کرد. تبدیل اصلی یک حذف گوسی است. برای این منظور از روش حذف گوس استفاده می شود تا هر بار یک متغیر زمینه x_k را از داخل زمینه ها به خارج از زمینه ها و یک متغیر x_{n+r} از خارج از زمینه ها به درون زمینه ها انتقال داده شود.

۹. مراجع

[1] Schnier Th., (Evolutionary Computation,) School of Computer Science, University of Birmingham, 2001.

[2] Abella J., Gonzalez A., Liosa J. and Vera X., "Near-Optimal Loop Tiling by means of Cache Miss Equations and Genetic Algorithms", *Politecnica Catalunya University and Instutionen for Datalink Malardalens Hogskola Vasteras*, Spain and Swweden (2000).

[3] Allen R. and Kennedy K., (Optimizing Compilers for Modern Architectures), Morgan Kaufmann Publishers, 2001.

[4] Ahmad I. and Kwok Y. K., "On Parallelizing the Multiprocessor Scheduling Problem", *IEEE Transaction on Parallel and Distributed Systems*, Vol. 10, No. 4, pp. 414-432 (April 1999).

[5] Ahmed N., Mateev N. and Pingali K., "Tiling Imperfectly-nested Loop Nests", *IEEE*, pp. 1-14 (2000).

[6] Al-Mouhamed M. A., "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs", *IEEE Transaction Software Engineering*, Vol. 16, No. 12, pp. 1390-1401 (December 1990).

[7] Andronikos T., Kalathas M., Ciorba F. M., Theodoropoulos P. and Papakonstantinou G., "An Efficient Scheduling of Uniform Dependence Loops", *Computing Systems Laboratory, Department of Electrical and Computer Engineering*, National Technical University of Athens Zographou Campus, Athens, Greece, pp. 1-10 (2004).

[8] Athanasaki M., Sotiropoulos A., Tsoukalas G., Koziris N. and Tsanakas P., "Hyperplane Grouping and Pipelined Schedules: How to Execute Tiled Loops Fast on Clusters of SMPs", *Journal of Supercomputing*, Vol. 32, pp. 197-226 (2005).

[9] Baiard F., Guerri D., Mori P. and Ricci L., "Evaluation of a Virtual Shared Memory Machine by the Compilation of Data Parallel Loops", *IEEE*, pp. 1-8 (1999).

[10] Bak T., (Evolutionary Algorithms in Theory and Practice,) Oxford University, 1996.

[11] Banerjee U., (Loop Transformations for Restructuring Compilers The Foundations,) Kluwer Academic, 1993.

[12] Baxter J. and Patel J. H., "The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm", *Proceeding of International Conference on Parallel Processing*, Vol. 2, pp. 217-222 (August 1989).

[13] Beckmann C. J., "Micro Architecture Support for Dynamic Scheduling of Acyclic Task Graphs", *University of Illinois*, Urbana, pp. 1-8 (1991).

[14] Bik A. J. C. and Wijshoff H. A. G., "Implementation of Fourier-Motzkin Elimination", *Leiden University*, Netherlands, pp. 1-10 (1994).

[15] Boulet P., Dongarra D., Robert Y. and Vivien F., "Tiling for Heterogeneous Computing Platforms", pp. 1-19 (1998).

[16] Boyle M. F. P. and Knijnenburg P. M. W., "Efficient Parallelisation using Combined Loop and Data Transformations", *IEEE*, pp. 1-10 (1998).

[17] Brest J. and Zumer V., "A Comparison of the Static Task Graph Scheduling Algorithms", *Faculty of Electrical Engineering and Computer Science Smetanova*, Maribor, Slovenia (xxxx).

[18] Chen D. K., Yew P. Ch., "On Effective Execution of Non-Uniform DOACROSS Loops", *IEEE*, pp. 1-6 (February 1995).

[19] Chu Ch. P. and Carver D. L., "Reordering the Statements with Dependence Cycles to Improve the Performance of Parallel Loops", *IEEE*, pp. 322-328 (1997).

[20] Chu P. C. and Beasley J. E., "A Genetic Algorithm for the Set Partitioning Problem", *The Management School Imperial College*, pp. 1-6 (1995).

[21] Chua H. T., Pean D. L. and Chen Ch., "M-hopping Method: An Effective Loop Scheduling Scheme for Non-uniform Dependence Loops", *Department of Computer Science and Information Engineering*, Hsinchu, Taiwan, Republic of China, 1-9 (1999).

[22] Cociorva D., Wilkins J. W., Lam C., Baumgartner G. and Ramanujam J., "Loop Optimization for a class of Memory-Constrained Computations", *ICS'01 Sorrento, Italy, ACM*, pp. 103-113 (2001).

[23] Coffman E. G., (Computer and Job-Shop Scheduling Theory,) John-Wiley, 1976.

[24] Correa R. C., Ferreira A. and Rebreyend P., "Scheduling Multiprocessor Tasks with Genetic Algorithms", *IEEE Transaction on Parallel and Distributed Systems*, Vol. 10, No. 8, pp. 825-837 (1999).

[25] Darte A., Robert Y. and Vivien F., (Scheduling and Automatic Parallelization,) Birkhäuser, 2000.

[26] Dhodhi M. K. and Ahmad I., "A Multiprocessor Scheduling Scheme Using Problem-Space Genetic Algorithms", *Proceeding of IEEE International Conference on Evolutionary Computation* (1995).

[27] Eisenbeis C. and Sogno J. C., "A General Algorithm for Data Dependence Analysis", *in International Conference on Supercomputing*, Washington, July 19-23, pp 1-28 (1992).

[28] Garey M. R. and Johnson D. S., (Computers and Intractability: A Guide to the Theory of NP-Completeness,) W.H. Freeman and Company, 1979.

[29] Gen M. and Cheng R., (Genetic Algorithms & Engineering Design,) John Wiley & Sons, 1997.

[30] Goldberg D. E., (Genetic Algorithms in Search, Optimization, and Machine Learning,) Addison-Wesley, 1989.

[31] Goumas G., Athanasaki M. and Koziris N., "Code Generation Methods for Tiling Transformations", *Journal of Information Science and Engineering*, Vol. 18, pp. 667-691 (2002).

[32] Goumas G., Sotiropoulos A. and Koziris N. "Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping", *IEEE*, pp. 1-10 (2001).

[33] Griebl M., Faber P. and Lengauer Ch., "Space-time mapping and tiling: a helpful combination", *Concurrency and Computation: Practice and Experience*, John Wiley & Sons, pp. 221-246 (2004).

[34] Griebl M., Feautrier P. A., Lengauer C., "Index set splitting", *International Journal of Parallel Programming*, Vol. 28, No. 6, pp. 607-631 (2000).

[35] Güyaüler B. and Gümrah F., "Comparison of Genetic Algorithm with Linear Programming for the Optimization of an Underground Gas-Storage Field", *Middle East Technical University*, Ankara, Turkey 1995, pp. 1-7 (2000).

[36] Hodzic E. and Shang W., "On Time Supernode Shape", *IEEE Transactions on Parallel and Distribyted Systems*, Vol. 13, No. 12, pp. 1220-1233 (December 2002).

[37] Hou E. S. H., Ansari N. and Ren H., “A Genetic Algorithm for Multiprocessor Scheduling”, *IEEE Transaction on Parallel and Distributed Computing*, Vol. 5, No. 2, pp. 113-120 (February 1994).

[38] Hsu Ch. and Kremer U., “A Quantitative Analysis of Tile Size Selection Algorithms”, *Journal of Supercomputing*, pp. 1-28 (2000).

[39] Hsu Ch. and Kremer U., “A Quantitative Analysis of Tile Size Selection Algorithms”, *Journal of Supercomputing*, Vol. 27, pp. 279-294 (2004).

[40] Högstedt K., Carter L. and Ferrante J., “On the Parallel Execution Time of Tiled Loops”, *in Transactions on Parallel and Distributed Systems*, Vol. 14, No. 3, pp. 307-320 (March 2003).

[41] Huang T. C., Hso P.H. and Sheng T. N., "Efficient Run-Time Scheduling for Parallelizing Partially Parallel Loop", *IEEE*, pp. 397-403 (1997).

[42] Hwang J. J., Chow Y. C., Anger F. D. and Lee C. Y., “Scheduling Precedence Graphs in Systems with Inter-processor Communication Times”, *SIAM Journal on Computer*, Vol. 18, No. 2, pp. 244-257 (April 1989).

[43] Jacobson T. and Stubbendieck G., "Dependency Analysis of For-Loop Structures for Automatic Parallelization of C Code", *Mathematics and Computer Science Department South Dakota School of Mines and Technology*, pp. 1-13 (2002).

[44] Kandemir M., Bordawekar R., Choudhary A. and Ramanujam J., “A Unified Tiling Approach for Out-O-Core Computations”, pp. 1-12 (1997).

[45] Kasahara H. and Narita S., “Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing”, *IEEE Transaction on Computers*, Vol. C-33, pp. 1023-1029 (November 1984).

[46] Kim S. J. and Browne J. C., "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures", *Proceeding Of International Conference on Parallel Processing*, Vol. 2, pp. 1-8 (August 1988).

[47] Kisuki T., Knijnenburg P.M.W. and O'Boyle M.F.P., "Combined Selection of Tile Sizes and Unroll Factors Using Iterative Compilation", *IEEE*, pp. 237-246 (2000).

[48] Kruatrachue B. and Lewis T. G., (Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems,) Technical Report, Oregon State University, Corvallis, OR 97331 (1987).

[49] Kwok Y. K. and Ahmad I., "Benchmarking and Comparison of the Task Graph Scheduling Algorithms", *Journal of Parallel and Distributed Computing 59*, pp. 381-422 (1999).

[50] Kwok Y. K. and Ahmad I., "Benchmarking and Comparison of the Task Graph Scheduling Algorithms", *Department of Electrical and Electronic Engineering*, University of Hong Kong, Pokfulam Road, Hong Kong (March 1999).

[51] Leopold C., "Exploiting Non-Uniform Reuse for Cache Optimization: A Case Study", *SAC, Las Vegas, ACM*, pp. 560-564 (2001).

[52] Manjikian N. and Abdelrahman T. S., "Scheduling of Wavefront Parallelism on Scalable Shared-memory Multiprocessors", *Department of Electrical and Computer Engineering*, University of Toronto, Toronto, Canada, pp. 1-10 (1996).

[53] Maydan D. E., Hennessy J. L. and Lam M. S., "Efficient and Exact Data Dependence Analysis", *ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, Toronto, Ontario, Canada, pp. 1-10 (June 26-28, 1991).

[54] Michalewicz Z., "A Survey of constraint Handling Techniques in Evolutionary Computation methods", *in McDonnell et al*, pp. 135-155 (1995).

[55] Michalewicz Z. and Attia N., "Evolutionary Computation of Constrained Problems", *in Sebald and Fogel*, pp. 98-108 (1994).

[56] Miyandashti F. J., (Loop Uniformization in Shared-Memory MIMD Machine,) Master Thesis, Iran University of Science and Technology, 1997 (Persian).

[57] Nakanishi T., Joe K., Polychronopoulos C. D., Araki K. and Fukuda A., "Estimating Parallel Execution Time of Loops with Loop-Carried Dependences", *in IEEE International Conference on Parallel Processing*, pp. 61-68 (1996).

[58] Obitko M., (Genetic Algorithms,) 1998.

[59] Parsa S, Lotfi Sh., Boushehrian O., Avaani A and Tasharrofi Sh, "On the use of Genetic Algorithms for Parallelization of Serial Loops in Super Compilers for Execution on Super Computers", *in 9th Annual Computer Society of Iran Computer Conference*, pp. 62-69 (2003) (Persian).

[60] Panda P. R., Nakamura H., Dutt N. D. and Nicolau A., "Augmenting Loop Tiling with Data Alignment for Improved Cache Performance", *IEEE Transactions on Computers*, Vol. 48, No. 2, pp. 142-149 (February 1999).

[61] Pean D. L., Chua H. T. and Chen CH., "A Release Combined Scheduling Scheme for Non-Uniform Dependence Loops", *Journal of Information Science and Engineering*, No. 18, pp. 223-255 (2002).

[62] Pike G. and N. Hilfenger P., "Better Tiling and Array Contraction for Compiling Scientific Programs", *IEEE*, pp. 1-12 (2002).

[63] Pit L. J., (Parallel Genetic Algorithms,) Master Thesis, Leiden University, 1995.

[64] Pugh W., *"The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependence Analysis"*, in Comm. of the ACM, pp. 1-18 (August 1992).

[65] Ramanujam J. and P. Sadayappan, "Tiling Multidimensional Iteration Spaces for Multicomputers", *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 108-120 (1992).

[66] Rastello F. and Robert Y., "Automatic Partitioning of Parallel Loops with Parallelepiped-Shaped Tiles", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 5, pp. 460-470 (May 2002).

[67] Rinehart M., Kianzad V. and Bhattacharyya Sh. S., "A Modular Genetic Algorithm for Scheduling Task Graphs, *Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies*, University of Maryland (xxxx).

[68] Rivera G. and Tseng CH. W., "Tiling Optimizations for 3D scientific Computations", *in the Proceeding of SC'00, Dallas, IEEE*, pp.1-23 (November 2000).

[69] Saffari M. and Bahrampour K., (Matrices and Vectors, Schwartz,) 1969 (Persian).

[70] Sarkar V., (Partitioning and Scheduling Parallel Programs for Multiprocessors,|) MIT Press, Cambridge, MA, 1989.

[71] Sarkar V. and Megiddo N., "An Analytical Model for Loop Tiling and its Solution", *IEEE*, pp. 146-153 (2000).

[72] Sathe S. R. and Nawghare P. M., "On Optimal Tiling of Iteration Spaces", *IEEE*, pp. 256-258 (2000).

[73] Sian Ch. F., “a Java based Distributed Approach to Genetic Programming on the Internet”, *University of Birmingham School of Computer Science* (1999).

[74] Sih G. C. and Lee E. A., “A Compile-Time Scheduling Heuristic for Interconnection Constrained Heterogeneous Processor Architectures”, *IEEE Transaction on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 75-87 (February 1993).

[75] Shirazi B., Wang M. and Pathak G., “Analysis and Evaluation of Heuristic Methods for Static Scheduling”, *Journal of Parallel and Distributed Computing*, No. 10, pp. 222-232 (1990).

[76] Song L. and Kavi K. M., "A Technique for Variable Dependence Driven Loop Peeling", *in the IEEE Proceeding of Fifth International Conference on Algorithms and Architectures for Parallel Processing*, pp. 1-6 (2002).

[77] William O’Neil T., “Techniques for Optimizing Loop Scheduling”, *PhD. Thesis*, the Graduate School of the University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, Indiana (April 2002).

[78] Wolf M. E., “Iteration space tiling for memory hierarchies”, *In Gary Rodrigue, the 3rd Conference on Parallel Processing for Scientific Computing*, pp. 357-361 (December 1989).

[79] Wolf M. E., “More iteration space tiling”, *Supercomputing'88*, pp. 655-664 (November 1989).

[80] Wolf M. E. and Lam M. S., “A Data Locality Optimizing Algorithm”, *ACM SIGPLAN’91 Conference on Programming Language Design and Implementation*, Toronto, Ontario, Canada, pp. 30-44 (June 26-28, 1991).

[81] Wolf M. E. and Lam M. S. “A Loop Transformation Theory and an Algorithm to Maximize Parallelism”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452-471 (October 1991).

[82] Wolf M. E., Maydan D. and Chen D. K., “Combining loop transformations considering caches and scheduling”, *29th Annual International Symposium on Micro-architecture*, pp. 274-286 (December 2-4, 1996).

[83] Wu A. S., Yu H., Jin Sh., Lin K. Ch. and Schiavone G., “An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling”, *IEEE Transaction on Parallel and Distributed Systems*, Vol. 15, No. 9, pp. 824-834 (September 2004).

[84] Wu Ch. T., Yang Ch. T. and Tseng Sh. Sh., "PPD: A Practical Parallel Loop Detector for Parallelizing Compilers", *IEEE*, pp. 274-281 (1996).

[85] Wu M. Y. and Gajski D. D., “Hypertool: A Programming Aid for Message-Passing Systems”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, pp. 330-343 (July 1990).

[86] Wu M., “MCP Revisited”, *Department of Electrical and Computer Engineering*, University of New Mexico (xxxx).

[87] Wu M. Y. and Gajski D. D., “Hypertool: A Programming Aid for Message-Passing Systems”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, pp. 330-343 (July 1990).

[88] Xue J., Huang Q. and Guo M., “Enabling Loop Fusion and Tiling for Cache Performance by Fixing Fusion-Preventing Data Dependences”, *International Conference on Parallel Processing*, pp. 1-9 (2005).

[89] Xue J., "Constructing DO Loops for Non-Convex Iteration Spaces in Compiling for Parallel Machines", *IEEE*, pp. 364-370 (1995).

[90] Yang Ch. T. and Cheng. K., “An Enhanced Parallel Loop Self-Scheduling Scheme for Cluster Environments”, *Journal of Supercomputing*, Vol. 34, pp. 315-335 (2005).

[91] Yang Ch. T., Tseng Sh. Sh., Hsieh M. H., Kao Sh. H. and Jiang M. F., "Run-Time Parallelization for Partially Parallel Loops", *IEEE*, pp. 308-314 (1997).

[92] Zhao Y. and Kennedy K., “Scalarization Using Loop Alignment and Loop Skewing”, *Journal of Supercomputing*, Vol. 31, pp. 5-46 (2005).

[93] Zima H. and Chapman B., (Supercompilers for Parallel and Vector Computers,) Addison-Wesley, 1991.