کاشی‌بندی حلقه‌های تودرتو با در نظر گرفتن محلیت داده‌ها به‌منظور اجرای موازی بر روی پردازنده‌های چندهسته‌ای

سعید پارسا[۱]، دانشیار، محمد حمزه‌ئی[۲]، دانشجوی دکتری

۱- دانشکده مهندسی کامپیوتر - دانشگاه علم و صنعت ایران - تهران - ایران - parsa@iust.ac.ir
۲- دانشکده مهندسی کامپیوتر - دانشگاه علم و صنعت ایران - تهران - ایران - hamzei@iust.ac.ir

چکیده: در سال‌های اخیر صنعت ریزپردازنده به سمت طراحی و ساخت پردازنده‌های چندهسته‌ای حرکت کرده است. این بستر محاسباتی با کارایی بالا دارای دو جنبه اصلی است: تعدادی هسته محاسباتی و سلسله مراتب حافظه نهان. به‌منظور استفاده از این بستر در جهت افزایش کارایی برنامه‌ها نیاز به تکنیک‌های کامپایلری مناسب با در نظر گرفتن این دو جنبه در کنار هم است. کاشی‌بندی حلقه‌های تکرار یکی از اصلی‌ترین تبدیلات حلقه‌ای است که هم به‌منظور موازی‌سازی دانه‌درشت در جهت استفاده از چندپردازنده‌ها و هم به‌منظور بهبود محلیت داده‌ها در جهت استفاده از سلسله مراتب حافظه نهان به‌کار رفته است. مشکل، کاربرد همزمان موازی‌سازی حلقه‌ها و بهبود محلیت داده‌ها در حلقه‌های تکرار است. در این مقاله، روشی نوین برای زمان‌بندی کاشی‌ها در جهت اجرای موازی کاشی‌ها بر اساس میزان استفاده مجدد داده‌ها بین آن‌ها ارائه شده است. در این روش بهبود محلیت داده‌ها با درنظر گرفتن سلسله مراتب حافظه نهان همگام با موازی‌سازی دانه‌درشت حاصل می‌شود.

واژه‌های کلیدی: کاشی‌بندی فضای تکرار، موازی‌سازی حلقه‌های تودرتو، بهبود محلیت داده‌ها، زمان‌بندی کاشی‌ها.

Nested Loops Tiling Considering Data Locality for Parallel Execution on Multi-core Processors

Saeed Parsa, Associate Professor[1], Mohammad Hamzei, PhD Candidate[2],

1- Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, parsa@iust.ac.ir
2- Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, hamzei@iust.ac.ir

Abstract: Recently, microprocessor industry has moved toward multi-core processor design and implementation. These high performance computing platforms have two important aspects: multiple computational cores and a hierarchy of cache. In order to use these platforms to speedup programs, there should be compiler techniques which consider these two aspects, simultaneously. An appropriate compiler technique is iteration space tiling, which is applied to obtain coarse grain parallelization as well as improving data locality in the nested loops on the multiprocessors. The problem is considering the loop parallelization and the data locality improvement, simultaneously. In this paper, a novel method for tile scheduling is presented in order to obtain parallel tiles based on the data reuse amongst them. Based on the proposed method, the data locality improvement according to the cache hierarchy along with the coarse grain parallelization is obtained.

Keywords: Iteration space tiling, nested loops parallelization, data locality improvement, tile scheduling.

تاریخ ارسال مقاله: ۹۲/۱۲/۲۸
تاریخ اصلاح مقاله: ۹۳/۰۶/۱۳
تاریخ پذیرش مقاله: ۹۳/۰۶/۲۳
نام نویسنده مسئول: سعید پارسا
نشانی نویسنده مسئول: ایران - تهران - نارمک - دانشگاه علم و صنعت ایران - دانشکده مهندسی کامپیوتر

۱- مقدمه

در گذشته افزایش سرعت برنامه‌ها با افزایش سرعت پردازنده‌ها و همچنین بهبود معماری پردازنده‌ها (مانند استفاده از سطوح حافظه نهان) در کنار استفاده از روش‌های بهینه‌سازی کامپایلری صورت می‌گرفت. در سال‌های اخیر صنعت ریزپردازنده به یک محدودیت فیزیکی که مشکل گرمایشی و توان مصرفی پردازنده‌ها است برخورد کرده است. باوجوداین محدودیت‌ها سرعت پالس ساعت پردازنده‌ها و درنتیجه سرعت پردازنده‌ها تقریباً بیش از این قابل‌افزایش نیست. افزایش تعداد هسته‌های محاسباتی بر روی یک تراشه اخیراً به‌عنوان راهکاری برای افزایش کارایی بدون ایجاد مشکل گرمایشی موردتوجه قرار گرفته است. امروزه پردازنده‌های چندهسته‌ای[۱] به‌عنوان بستر محاسباتی در کاربردهای مختلف از کاربردهای خانگی تا محاسبات علمی و ابرمحاسبات به کار گرفته می‌شود [۱].

بااین‌وجود، این چندپردازنده‌ها به‌صورت مستقیم کارایی برنامه‌های ترتیبی را بهبود نمی‌دهند. به‌منظور افزایش کارایی برنامه‌ها در این بسترهای سخت‌افزاری نیاز به تکنیک‌های نرم‌افزاری مناسب احساس می‌شود. ازآنجاکه ایجاد برنامه‌های موازی مشکل‌تر از برنامه‌های ترتیبی است، وجود ابزارهای کامپایلری به‌منظور استخراج توازی و تولید خودکار کد موازی از کد ترتیبی بسیار موردتوجه قرار گرفته است. همچنین با توجه به اینکه حجم پردازشی اصلی در برنامه‌های علمی و محاسباتی در حلقه‌ها انجام می‌شود، بیشتر تحقیقات کامپایلری درزمینه موازی‌سازی خودکار حلقه‌های تودرتو صورت می‌گیرد. یک سؤال مطرح درزمینه کامپایلر برای سیستم‌های چندپردازنده‌ای این است که چه تبدیلات حلقه‌ای و با چه ترتیبی باید اعمال شود تا افزایش سرعت برنامه‌ها متناسب با تعداد هسته‌های محاسباتی حاصل شود.

به‌منظور به دست آوردن تبدیلات مناسب باید دو جنبه اصلی این معماری‌ها یعنی وجود هسته‌های محاسباتی متعدد و همچنین سلسله‌مراتب حافظه نهان[۲] آن‌ها در نظر گرفته شود. برای استفاده از هسته‌های محاسباتی مختلف باید تبدیلاتی در جهت موازی‌سازی برنامه‌ها و به‌منظور استفاده از سلسله‌مراتب حافظه باید تبدیلاتی در جهت افزایش استفاده مجدد از داده‌ها[۳] در سطوح داخلی‌تر حافظه نهان و درنتیجه بهبود محلیت[۴] داشته باشیم. مسئله‌ای که وجود دارد این است که موازی‌سازی باهدف توزیع حلقه‌ها بر روی پردازنده‌های مختلف تا حدی در تضاد با بهبود محلیت است که در جهت متمرکزسازی داده‌ها برای استفاده مجدد است [۲]. علیرغم وجود این تضاد، کاشی‌بندی فضای تکرار حلقه‌ها تبدیلی است که به‌منظور رسیدن به این دو هدف به‌صورت مجزا به‌کار رفته است [۳]. کاشی‌بندی فضای تکرار[۵] از طرفی سعی در متمرکز ساختن محاسبات بر روی بخشی از داده‌ها در قالب یک بلاک دارد. از طرف دیگر با انجام کاشی‌بندی، فضای تکرار به بلاک‌هایی تقسیم می‌شود که با زمان‌بندی مناسب می‌توانند به‌صورت موازی اجرا شوند.

روش موجود به‌منظور زمان‌بندی کاشی‌ها برای اجرای موازی روش امواج[۶] است [۴، ۵]. در این روش تمام کاشی‌هایی که بر روی یک موج قرار می‌گیرند به‌صورت موازی اجرا می‌شوند. در این روش فقط موازی‌سازی کاشی‌ها موردتوجه بوده است و عامل مهم دیگر یعنی محلیت داده‌ها در نظر گرفته نشده است. همچنین تعداد کاشی‌های قرارگرفته بر روی امواج مختلف متفاوت بوده و درنتیجه کد حاصل دارای توازن بار بر روی پردازنده‌ها نیست.

در این مقاله یک رویکرد کارا به‌منظور کاشی‌بندی فضای تکرار حلقه‌های تودرتو و زمان‌بندی کاشی‌ها به‌منظور اجرای موازی بر روی هسته‌های پردازشی مختلف با در نظر گرفتن بهبود محلیت داده‌ها ارائه می‌شود. در روش ارائه‌شده، حلقه‌های تودرتو در مدل چندوجهی[۷] در قالب چندوجهی‌های فضای تکرار و وابستگی نمایش داده می‌شود. سپس تبدیل مناسب در جهت به دست آوردن فضای تکرار قابل کاشی‌بندی مستطیلی به دست می‌آید. بر اساس تبدیل به‌دست‌آمده و با در نظر گرفتن اندازه حافظه نهان و میزان ارتباطات بین کاشی‌ها، اندازه و شکل دقیق کاشی‌ها به دست می‌آید. سپس یک گراف وزن‌دار که نشان‌دهنده کاشی‌ها و میزان ارتباطات بین آن‌ها است از فضای کاشی‌بندی شده استخراج می‌شود. درنهایت بر اساس یک الگوریتم کارا، زمان‌بندی این گراف با در نظر گرفتن تعداد هسته‌های پردازشی و سلسله‌مراتب حافظه نهان پردازنده صورت می‌گیرد. نکته حائز اهمیت این است که زمان‌بندی کاشی‌ها مستقل از کاشی‌بندی است و می‌توان از هر الگوریتم دیگری به‌منظور کاشی‌بندی فضای تکرار استفاده کرد.

نوآوری‌های این مقاله عبارت‌اند از:

- ارائه یک چارچوب به‌منظور موازی‌سازی حلقه‌های تکرار و بهبود محلیت داده‌های دسترسی‌شده شامل مؤلفه‌های (۱) ایجادکننده فضای قابل کاشی‌بندی (۲) کاشی‌بند و (۳) زمان‌بند ایستای کاشی‌ها
- ارائه یک الگوریتم نوین زمان‌بندی ایستای کاشی‌ها که با در نظر گرفتن سلسله‌مراتب حافظه نهان پردازنده، محلیت داده‌ها در سطوح مختلف حافظه نهان اختصاصی[۸] و اشتراکی[۹] و توازی همراه با توازن بارکاری برای هسته‌های پردازشی مختلف را نتیجه می‌دهد

بخش‌های بعدی مقاله به‌این‌ترتیب سازمان‌دهی شده است: در بخش ۲ کارهای مرتبط انجام‌شده موردبحث قرار گرفته است. بخش ۳ به معرفی مدل چندوجهی و همچنین مقدمات موردنیاز برای ارائه روش پیشنهادی می‌پردازد. در بخش ۴ مراحل انجام تبدیلات به‌منظور ایجاد فضای قابل کاشی‌بندی و همچنین کاشی‌بندی و زمان‌بندی کاشی‌ها ارائه می‌شود. بخش ۵ نتایج تجربی استفاده از روش پیشنهادی را بر روی برنامه‌های محک مختلف نشان داده است. درنهایت نتیجه‌گیری بیان شده است.

۲- کارهای مرتبط

در این بخش کارهای مرتبط درزمینه موازی‌سازی حلقه‌ها و بهبود محلیت داده‌ها را مورد بررسی قرار می‌دهیم. اغلب کارهای مرتبط انجام‌شده عموماً یکی از این دو فاکتور را در انجام بهینه‌سازی مدنظر قرار داده‌اند [۶،۷ و ۸]. درصورتی‌که همان‌طور که در [۳] بررسی شده است در نظر گرفتن یکی از این دو فاکتور به‌منظور بهینه‌سازی برنامه‌ها برای اجرا بر روی چندپردازنده‌ها در حالت کلی منجر به حالت بهینه نمی‌شود. در نظر گرفتن این دو فاکتور در کنار هم به‌منظور افزایش کارایی برنامه‌ها اخیراً موردتوجه قرار گرفته است. در ادامه کارهای اصلی انجام‌شده درزمینه موازی‌سازی و بهبود محلیت در کنار هم را مورد بحث قرار می‌دهیم.

در [۵] یک روش کارا با در نظر گرفتن این دو فاکتور در کنار هم ارائه شده است. مشکل اصلی روش ارائه‌شده این است که به دلیل عدم استفاده از مدل چندوجهی در جهت نمایش و بهینه‌سازی حلقه‌ها، برای حالات خاص حلقه‌ها قابل استفاده است و درصورتی‌که حلقه‌ها کامل نباشند قابل استفاده نیست. تبدیلات در نظر گرفته‌شده در این روش فقط تبدیلات تک‌پیمانه‌ای[۱۰] است. همچنین از روش امواج در جهت زمان‌بندی اجرای کاشی‌ها استفاده شده است که در آن مسئله بهبود محلیت در سطح کاشی‌ها در نظر گرفته نمی‌شود و منجر به عدم توازن بار کاری در زمان اجرا می‌شود.

در [۲] یک روش استخراج توازی در کنار بهبود محلیت با ایجاد یک شبکه محدودیت ارائه شده است و درنهایت با استفاده از الگوریتم‌های مکاشفه‌ای به دنبال پیدا کردن ترکیبی از تبدیلات مناسب است. در این روش فقط تبدیلات خطی مدنظر قرار گرفته است و درنتیجه کاشی‌بندی به‌عنوان تبدیل مناسب در جهت استخراج توازی دانه‌درشت همراه با بهبود محلیت در نظر گرفته نشده است.

تا جایی که اطلاع داریم پلوتو[۱۱] [۳] اولین موازی‌ساز خودکار حلقه‌های تودرتو است که بر مبنای حالت‌های خاص حلقه‌ها نبوده (مثلاً حلقه‌های کامل با وابستگی‌های هم‌شکل[۱۲]) و بهبود محلیت را در نظر می‌گیرد. در روش استفاده‌شده در پلوتو کاشی‌بندی فضای تکرار بر مبنای کم کردن حدی بر روی فاصله وابستگی‌ها بوده و پس از کاشی‌بندی، زمان‌بندی و تخصیص کاشی‌ها به پردازنده‌ها برای اجرای موازی به‌صورت پویا و در زمان اجرا صورت می‌گیرد. مشکل اصلی این روش عدم توجه به استفاده مجدد داده‌ها به‌منظور تخصیص کاشی‌ها به پردازنده‌ها است. همچنین با توجه به زمان‌بندی پویا که در زمان اجرا انجام می‌شود، این روش دارای سربار محاسباتی است. درواقع تمرکز اصلی پلوتو بر روی کاشی‌بندی فضای تکرار است درصورتی‌که پس از کاشی‌بندی، زمان‌بندی ایستای کاشی‌ها بدون ایجاد سربار زمان اجرا می‌تواند بهبود محلیت را با بهره‌برداری از استفاده مجدد داده‌های دسترسی‌شده درون کاشی‌های مختلف و بهبود توازی را با تخصیص دارای توازن بارکاری به هسته‌های پردازشی نتیجه دهد. همچنین در [۹] یک رویکرد جدید بر مبنای سطح ارضای وابستگی‌های داده‌ای به‌منظور ایجاد فضای تکرار قابل کاشی‌بندی با در نظر گرفتن استخراج توازی و بهبود محلیت داده‌ها برای حلقه‌های تکرار ارائه شده است. توجه شود که روش ارائه‌شده در مقاله حاضر به‌منظور زمان‌بندی کاشی‌ها است و پیش‌نیاز آن قابل کاشی‌بندی بودن فضای تکرار است. از روش ارائه‌شده در [۹] می‌توان برای کاشی‌بندی فضای تکرار استفاده نمود.

در [۱۰] یک روش زمان‌بندی بر اساس روش پلوتو ارائه شده است که در آن از یک الگوریتم خوشه‌بندی[۱۳] به‌منظور به دست آوردن خوشه‌هایی با قابلیت اجرای موازی ارائه شده است. پس از خوشه‌بندی، این خوشه‌ها برای اجرای موازی زمان‌بندی می‌شوند. مشکل این روش این است که با اجرای هر یک از کاشی‌های درون یک خوشه، ممکن است وابستگی‌های کاشی‌های دیگری ارضا شود که ازنظر محلیت داده‌ای اولویت بیشتری برای اجرا داشته باشند. در این مقاله از معیار بهینه‌سازی متفاوتی در جهت افزایش توازی دانه‌درشت[۱۴] استفاده شده است و تخصیص و زمان‌بندی کاشی‌ها به‌صورت ایستا و بر مبنای سلسله‌مراتب حافظه نهان چندپردازنده با در نظر گرفتن استفاده مجدد داده‌های موجود در حافظه نهان و در نظر گرفتن توازن بار تقریبی انجام می‌شود. نتایج ارزیابی نشان‌دهنده کارایی بالای روش پیشنهادی در جهت افزایش توازی دانه‌درشت و بهبود محلیت داده‌ها در جهت افزایش سرعت اجرایی برنامه‌ها است.

۳- مفاهیم اولیه

در این بخش مفاهیم اولیه موردنیاز شامل مدل چندوجهی برای نمایش حلقه‌ها، مفهوم تبدیلات و زمان‌بندی و همچنین کاشی‌بندی فضای تکرار ارائه می‌شود.

۳-۱- مدل چندوجهی

مدل چندوجهی یک نمایش انعطاف‌پذیر و صریح برای نمایش حلقه‌ها بر مبنای جبرخطی ارائه می‌دهد. در مدل چندوجهی برخلاف مدل‌های دیگر نمایش برنامه‌ها ازجمله درخت نحوی، به‌جای دستورات، نمونه دستورات در نظر گرفته می‌شوند. درنتیجه در این مدل انواع تبدیلات مختلف در قالب جابه‌جایی‌هایی از نمونه‌های مختلف دستورات نمایش داده می‌شوند. بدین ترتیب می‌توان توالی تعدادی از تبدیلات مختلف بر روی برنامه را که درنهایت منجر به یک ترتیب اجرایی برای نمونه دستورات می‌شود در قالب یک تبدیل در مدل چندوجهی نشان داد. در مدل چندوجهی برای نمایش حلقه‌های تودرتو از چندوجهی دامنه دستورات برای مشخص کردن فضای تکرار دستورات درون حلقه‌ها و از چندوجهی وابستگی برای نمایش وابستگی بین نمونه دستورات درون حلقه‌ها استفاده می‌شود [۱۱].

دامنه یک دستور نشان‌دهنده نمونه‌های پویای آن دستور است که به‌صورت یک چندوجهی و در قالب مجموعه‌ای از نامساوی‌های خطی

به‌صورت (۱) نمایش داده می‌شود. هر نقطه درون این چندوجهی مشخص‌کننده یک اجرای دستور مربوطه است.

$$\mathbf{D}_s.(X_s,n,1)^{\mathrm{T}} \geq 0 \qquad (۱)$$

که در آن $\mathbf{D}_s$ ماتریسی است که محدودیت‌های مربوط به حدود حلقه‌ها را نشان می‌دهد. X_s اندیس حلقه‌های دربرگیرنده و n بردار پارامترهای برنامه است.

وابستگی‌ها در یک برنامه کنترلی ایستا می‌توانند در قالب چندوجهی‌های وابستگی به‌صورت فرمال بیان شود. یک چندوجهی وابستگی $\mathbf{D}_{r,s}$ یک زیرمجموعه از ضرب کارتزین دامنه تکرار دستورات r و s است. هر نقطه صحیح درون چندوجهی وابستگی متناظر با جفتی از نمونه دستورات وابسته است. مدل چندوجهی مربوط به یک نمونه حلقه تودرتو در مثال ۱ نشان داده شده است.

مثال ۱. به‌عنوان‌مثال چندوجهی دامنه دستورات و همچنین چندوجهی وابستگی برای حلقه زیر مشخص شده است:

For i=0 to N
For j=0 to M
Stmt: A[i][j]=A[i-1][j];

$$\mathbf{D}_{stmt} = \{(i,j) \mid 0 \leq i \leq N \wedge 0 \leq j \leq M\}$$

$$\mathbf{D}_{stmt,stmt} = \{(\mathrm{i,j,i',j'}) \mid 1 \leq \mathrm{i} \leq \mathrm{N} \wedge 1 \leq \mathrm{j} \leq \mathrm{M} \wedge 1 \leq \mathrm{i'} \leq \mathrm{N} \wedge 1 \leq \mathrm{j'} \leq \mathrm{M} \wedge \mathrm{i} = \mathrm{i'}-1 \wedge \mathrm{j} = \mathrm{j'}\}$$

همان‌طور که مشخص است $\mathbf{D}_{stmt}$ چندوجهی دامنه دستور *stmt* است که مشخص‌کننده فضای تکرار برای این دستور است. همچنین چندوجهی وابستگی $\mathbf{D}_{stmt,stmt}$ مشخص‌کننده تمام نمونه دستوراتی از دستور *stmt* است که به نمونه دستوری از همان دستور وابسته است. به‌عنوان‌مثال ازآنجاکه نقطه (0,0,1,0) درون این چندوجهی قرار دارد پس نمونه دستور *stmt* در تکرار (1,0) وابسته به نمونه دستور *stmt* در تکرار (0,0) است.

۳-۲- تبدیلات حلقه‌ای

همان‌طور که گفته شد چندوجهی دامنه دستورات مشخص‌کننده مجموعه نمونه‌های دستورات است. ولی این چندوجهی ترتیب اجرایی نمونه دستورات را مشخص نمی‌کند. به‌منظور مشخص کردن ترتیب اجرایی نمونه دستورات از یک تابع زمان‌بندی برای هر دستور استفاده می‌شود. این تابع زمان‌بندی، زمان منطقی اجرای هر نمونه دستور را برای دستور مربوطه بر اساس شمارنده‌های حلقه‌های دربرگیرنده و پارامترهای برنامه مشخص می‌کند [۱۱]. برای هر دستور *stmt*، زمان اجرای نمونه دستور $X_s=(i,j,\ldots)$ به‌صورت (۲) است ($i,j,\ldots$ اندیس‌های حلقه‌های دربرگیرنده دستور *stmt* هستند):

$$Time(X_s) = \mathbf{T}_s.(X_s,p,1)^T \qquad (۲)$$

که در آن $\mathbf{T}_s$ ماتریس زمان‌بندی با ابعاد $m*(n+p+1)$ از ثوابت است. m مشخص‌کننده تعداد سطرهای ماتریس زمان‌بندی، n تعداد حلقه‌های دربرگیرنده دستور و p تعداد پارامترهای ورودی است. به‌عنوان‌مثال برای حلقه مثال ۱ تابع زمان‌بندی که مشخص‌کننده ترتیب اصلی اجرای نمونه دستورات *stmt* است با ماتریس زیر مشخص می‌شود:

$$\mathbf{T}_s = \begin{bmatrix} 10000 \\ 01000 \end{bmatrix}$$

که در آن هر سطر ماتریس مشخص‌کننده ضرایب $X_s=(i,j,\mathrm{N,M,1})$ است.

هر تبدیل و اعمال آن بر روی فضای تکرار یک ترتیب اجرایی جدید را نتیجه می‌دهد. درنتیجه هر تبدیل را می‌توان با یک تابع زمان‌بندی نشان داد که مشخص‌کننده یک ترتیب جدید اجرایی است [۱۲]. با توجه به وجود وابستگی‌ها، هر ترتیب اجرایی و درنتیجه هر زمان‌بندی برای دستورات مجاز نیست. دو نمونه دستور وابسته هستند اگر به مکان حافظه یکسانی دسترسی پیدا کنند و یکی از آن‌ها نوشتن باشد. هر تبدیلی که ترتیب اجرایی نمونه‌های دستورات وابسته به هم را حفظ کند یک تبدیل مجاز است. درنتیجه با در نظر گرفتن اطلاعات وابستگی داده‌ای می‌توان شرایط مجاز بودن تبدیلات را به دست آورده و با جستجوی این فضای تبدیلات مجاز بر اساس یک معیار مناسب، بهینه‌سازی موردنظر را انجام داد [۱۳،۱۴]. هدف موردنظر ما در اینجا به دست آوردن یک فضای تکرار قابل کاشی‌بندی است که در بخش بعد موردبررسی قرار گرفته است.

اعمال یک ماتریس زمان‌بندی $\mathbf{T}_{stmt}$ بر روی دستور *stmt* مجاز است اگر و تنها اگر

$$\forall e \in E \forall \langle x_1,x_2 \rangle \in \rho_e \mathbf{T}_{Stmt2}(x_2) - \mathbf{T}_{Stmt1}(x_1) \geq 0 \qquad (۳)$$

بر اساس این نامعادله، اعمال ماتریس زمان‌بندی بر روی نمونه دستور x_2 از دستور *stmt2* که به نمونه دستور x_1 از *stmt1* وابسته است (زیرا $<x_1,x_2>$ درون چندوجهی وابستگی P_e قرار دارد)، باید یک زمان منطقی بزرگ‌تر برای x_2 نسبت به x_1 نتیجه دهد. درواقع بر اساس این معادله مقصد وابستگی باید بعد از مبدأ وابستگی اجرا شود.

۳-۳- کاشی‌بندی فضای تکرار

با کاشی‌بندی فضای تکرار، فضای تکرار دسترسی‌شده به بلاک‌های کوچک‌تری تقسیم می‌شود و درنتیجه آرایه‌های چندبعدی دسترسی‌شده در حلقه‌های تودرتو که خیلی بزرگ هستند و در حافظه نهان جا نمی‌شوند به‌صورت منطقی به بخش‌های کوچک‌تری تقسیم‌بندی می‌گردند که در حافظه نهان جا شوند [۱۵]. به‌منظور به دست آوردن فضای تکرار قابل کاشی‌بندی از قضیه زیر استفاده می‌کنیم.

قضیه ۱. حلقه‌های $\mathrm{I_i}$ تا $\mathrm{I_j}$ می‌توانند کاشی‌بندی شوند اگر این حلقه‌ها جابه‌جاپذیر کامل[۱۵] باشند [۵].

تعریف ۱. دسته جابه‌جاپذیر کامل از حلقه‌ها: هنگامی‌که همه مؤلفه‌های وابستگی‌ها غیرمنفی باشند، هر جابه‌جایی حلقه‌ها منجر به وابستگی‌های تبدیل یافته مثبت الفبایی می‌شود و درنتیجه مجاز است. در این حالت گفته می‌شود که حلقه‌ها جابه‌جاپذیر کامل هستند.

درنتیجه به‌منظور به دست آوردن فضای تکرار قابل کاشی‌بندی، باید تبدیلاتی در جهت به دست آوردن حلقه‌های جابه‌جاپذیر کامل انجام شود. چنین تبدیلاتی دارای ویژگی زیر هستند.

اعمال سطوح $p,p+1,...,p+s$ از ماتریس زمان‌بندی $\mathbf{T}_s$ یک دسته از حلقه‌های جابه‌جاپذیر را در فضای تبدیل یافته تشکیل می‌دهند اگر و تنها اگر:

$$\forall e \in E_p \forall \langle x_1, x_2 \rangle \in \rho_e \forall k : p \le k \le p+s$$
$$\mathbf{T}_{stmt2}{}^k(x_2) - \mathbf{T}_{stmt1}{}^k(x_1) \ge 0 \quad (۴)$$

که در آن $\mathbf{T}_{stmt}{}^k(x_s)$ سطر kام ماتریس زمان‌بندی است و E_p مجموعه وابستگی‌هایی است که تا سطح p-1 ارضا نشده‌اند. در بخش بعدی نحوه به دست آوردن فضای تکرار قابل کاشی‌بندی مستطیلی با استفاده از این مفاهیم و رویکرد پیشنهادی در جهت زمان‌بندی کاشی‌ها بیان شده است.

۴- رویکرد پیشنهادی

در این بخش رویکرد پیشنهادی در جهت تغییر ساختار حلقه‌های تکرار به‌منظور اجرا بر روی سیستم‌های چندهسته‌ای بیان می‌شود. ابتدا به دنبال یک مجموعه شامل دسته‌هایی از حلقه‌های جابه‌جاپذیر کامل هستیم. سپس کاشی‌بندی مستطیلی فضای تکرار به‌دست‌آمده انجام می‌شود و زمان‌بندی کاشی‌ها برای اجرای موازی صورت می‌گیرد.

بهترین حالت زمانی است که بیشترین حلقه را در بیرونی‌ترین دسته جابه‌جاپذیر کامل داشته باشیم، زیرا در این صورت می‌توان با کاشی‌بندی این حلقه‌ها توازی دانه‌درشت‌تری را استخراج کرد. برای رسیدن به این هدف سطرهای ماتریس تبدیلات را تا جایی که وابستگی‌ها اجازه دهند به ترتیب به نحوی به دست می‌آوریم که شرایط لازم برای به دست آوردن دسته‌ای از حلقه‌های جابه‌جاپذیر کامل را داشته باشند. برای این منظور ابتدا تمام نامعادلات مربوط به محدودیت‌های مجاز بودن تبدیلات (رابطه ۳) برای هر وابستگی به دست می‌آیند. این مجموعه از نامعادلات تشکیل یک دستگاه نامعادلات می‌دهند. سپس به دنبال جوابی در فضای ضرایب زمان‌بندی برای این نامعادلات هستیم (توجه شود که این دستگاه حداقل یک جواب دارد که منجر به تبدیل همانی می‌شود که حلقه اصلی را نتیجه می‌دهد). هر جواب مستقل این نامعادلات یک سطر از ماتریس زمان‌بندی را تشکیل می‌دهند. هنگامی‌که جواب مستقل دیگری یافت نشود، نامعادلات مربوط به وابستگی‌های ارضا شده توسط جواب‌های یافته‌شده را از دستگاه نامعادلات حذف می‌کنیم (درواقع وابستگی‌های ارضا شده توسط سطوح قبلی ماتریس زمان‌بندی حذف می‌شوند) و به‌صورت بازگشتی، دسته جواب‌های بعدی به دست می‌آیند تا درنهایت تمام وابستگی‌ها ارضا شوند. با توجه به قضیه ۱ هر دسته از جواب‌ها، تشکیل یک دسته از حلقه‌های جابه‌جاپذیر کامل را در فضای تکرار به‌دست‌آمده می‌دهند که به‌صورت مستطیلی قابل کاشی‌بندی است. ازآنجاکه با اعمال ماتریس زمان‌بندی به‌دست‌آمده، دسته‌هایی از حلقه‌های جابه‌جاپذیر کامل به دست می‌آید، می‌توان کاشی‌بندی مستطیلی فضای تکرار را در جهت محورهای فضای تکرار مقصد در این دسته‌ها انجام داد.

در کاشی‌بندی فضای تکرار، کاشی‌ها دو مشخصه اصلی دارند: شکل و اندازه. با توجه به الگوریتم مطرح‌شده برای به دست آوردن فضای تکرار قابل کاشی‌بندی، فضای تکرار به‌دست‌آمده با اعمال ماتریس تبدیل، قابل کاشی‌بندی به‌صورت مستطیلی در جهت محورهای تشکیل‌دهنده فضای تکرار است. با توجه به اینکه اندازه هر کاشی باید به نحوی محدود شود که داده‌های دسترسی شده در آن بتوانند درون حافظه نهان محلی قرار گیرند و از طرفی هر بعد کاشی به میزان متفاوتی از ارتباطات نیاز دارد، در این مقاله اندازه کاشی بر اساس فرم کاشی به دست می‌آید که به‌صورت نسبت ابعاد کاشی به یکدیگر تعریف می‌شود .

به‌منظور تعیین فرم کاشی، نسبت ابعاد کاشی را عکس نسبت میزان ارتباطات در آن بعد کاشی به سایر ابعاد در نظر می‌گیریم. به‌این‌ترتیب، بعدی که منجر به ارتباطات بیشتر می‌شود اندازه کوچک‌تری نسبت به بعدی با ارتباطات کمتر دارد. پس از تعیین فرم کاشی‌ها، اندازه دقیق هر بعد بر اساس گنجایش حافظه نهان محلی گره‌های محاسباتی تعیین می‌شود به‌نحوی‌که کل داده‌های مجزای دسترسی شده در هر کاشی بتواند درون حافظه نهان قرار گیرد.

پس از تعیین شکل دقیق کاشی‌ها، زمان‌بندی کاشی‌ها برای اجرا بر روی چندپردازنده‌ها، بر اساس میزان اشتراک داده بین آن‌ها و با در نظر گرفتن سلسله‌مراتب حافظه نهان چندپردازنده صورت می‌گیرد. میزان اشتراک داده‌ها بین دو کاشی معادل میزان وابستگی‌های داده‌ای با در نظر گرفتن وابستگی‌های RAR (Read-After-Read) است.

به‌منظور زمان‌بندی کاشی‌ها، در مرحله اول بر اساس کاشی‌های به‌دست‌آمده و ارتباطات بین آن‌ها یک گراف جهت‌دار وزن‌دار استخراج می‌شود. گره‌های این گراف را کاشی‌ها تشکیل می‌دهند و درصورتی‌که دو کاشی به هم وابسته باشند یک لبه بین آن‌ها قرار می‌گیرد و تعداد وابستگی‌های داده‌ای (با در نظر گرفتن وابستگی‌RAR) که معیار دقیقی برای میزان استفاده مجدد داده‌ها بین آن‌ها است را وزن یال مربوطه در نظر می‌گیریم. همچنین هر لبه یک برچسب دارد که مشخص‌کننده نوع وابستگی بین دو کاشی است که دو حالت دارد: وابستگی RAR و وابستگی غیر RAR. توجه شود که اطلاعات وابستگی‌های RAR به‌منظور اختصاص مناسب کاشی‌ها بر اساس میزان استفاده مجدد داده‌ها به پردازنده‌ها است و لزومی به رعایت این نوع وابستگی‌ها وجود ندارد ولی به‌منظور اجرای صحیح برنامه، سایر وابستگی‌ها حتماً باید رعایت شوند. در شکل (۱)، حلقه تودرتوی کاشی‌بندی شده مربوط به برنامه ضرب ماتریسی و گراف اشتراک داده‌ای مربوطه نشان داده شده است. برچسب یال‌های این گراف برابر با تعداد عناصر داده‌ای مشترک بین کاشی‌های مربوطه است. در این مثال، تمامی یال‌ها از نوع RAR هستند.

الگوریتم زمان‌بندی پیشنهادی نوعی از زمان‌بندی لیست است. در الگوریتم مطرح‌شده هر پردازنده لیستی از حافظه‌های نهان را به ترتیب از حافظه نهان اختصاصی خود تا درنهایت بیرونی‌ترین سطح حافظه نهان که حافظه نهان اشتراکی تمام پردازنده‌ها است به همراه کاشی‌های موجود در آن‌ها در هرلحظه نگه‌داری می‌کند. در اینجا فرض شده است که الگوریتم جایگذاری حافظه نهان LRU (least recently used) است.

به‌منظور زمان‌بندی کاشی‌ها یک جدول زمان‌بندی در نظر گرفته شده است که ستون‌های آن را پردازنده‌ها و سطرهای آن را سیکل‌های زمانی در نظر گرفته‌شده برای پردازنده‌ها تشکیل می‌دهند. منظور از سیکل زمانی برای هر پردازنده، یک سیکل مجازی است که در آن پردازنده می‌تواند محاسبات مربوط به یک کاشی را اجرا کند. نمونه‌ای از جدول زمان‌بندی در شکل (۱) نشان داده شده است. ازآنجاکه اندازه کاشی‌ها به‌جز کاشی‌های حاشیه‌ای برابر است، در اینجا فرض می‌کنیم هر کاشی در یک سیکل زمانی مجازی اجرا می‌شود. یک گره گراف در یک سیکل زمانی قابل زمان‌بندی است اگر مبدأ تمام یال‌های غیرRAR ورودی به آن در سیکل‌های قبل، زمان‌بندی شده باشد. همچنین ازآنجاکه یال‌های از نوع غیر RAR باعث ایجاد محدودیت در ترتیب اجرای کاشی‌ها می‌شوند، زمان‌بندی آن‌ها دارای اولویت بیشتری هستند زیرا با زمان‌بندی آن‌ها تمام کاشی‌های وابسته به آن‌ها به لیست کاشی‌های آزاد اضافه و قابل زمان‌بندی می‌شوند.

به‌منظور زمان‌بندی یک سیکل مجازی از پردازنده‌ها، ابتدا گره‌های قابل زمان‌بندی که حداقل مبدأ یک یال غیرRAR هستند زمان‌بندی می‌شوند. سپس در صورت وجود هسته پردازشی، سایر گره-ها زمان‌بندی می‌شوند. برای این منظور لیستی از گره‌های گراف که مبدأ حداقل یک یال غیرRAR هستند و قابل زمان‌بندی هستند در نظر گرفته می‌شود. سپس اولویت هر یک از این گره‌ها برابر با اندازه طولانی‌ترین مسیر از آن‌ها به یک گره پایانی در نظر گرفته می‌شود (اندازه مسیر برابر با تعداد گره‌های موجود در مسیر در نظر گرفته می‌شود). سپس تا زمانی که تمام گره‌های این لیست زمان‌بندی نشده‌اند هر بار گره با بیشترین اولویت انتخاب شده و میزان اشتراک داده‌ای آن با گره موجود در حافظه نهان اختصاصی هر یک از هسته‌های پردازشی با استفاده از گراف استفاده مجدد تعیین شده و گره برای اجرا بر روی هسته پردازشی با بیشترین اشتراک داده‌ای زمان‌بندی می‌شود. درصورتی‌که اشتراک داده‌ای بین گره با گره‌های موجود در حافظه نهان اختصاصی پردازنده وجود نداشته باشد، میزان اشتراک داده‌ای با سطح بعدی حافظه نهان بررسی می‌شود و این عمل مجدداً تکرار می‌شود. این عمل تا زمانی که تمام گره‌های از این نوع زمان‌بندی نشده‌اند و همچنین وجود سیکل جاری خالی برای هسته پردازشی ادامه می‌یابد.

سپس در صورت وجود سیکل خالی لیستی از سایر گره‌های قابل زمان‌بندی که مبدأ وابستگی غیرRAR نیستند در مقابل لیستی از پردازنده‌ها که دارای سیکل زمانی جاری خالی می‌باشند را در نظر گرفته و بر اساس آن یک گراف دوطرفه وزن‌دار تشکیل می‌دهیم. وزن هر یال از این گراف، که یک کاشی را به یک پردازنده متصل می‌کند، را برابر با میزان اشتراک داده‌ای بین آن کاشی و کاشی‌های موجود در حافظه نهان اختصاصی آن پردازنده در نظر می‌گیریم. سپس اولویت یال‌ها را برابر میزان اشتراک داده‌ای کاشی مربوطه با کاشی موجود در حافظه نهان اختصاصی هسته‌های پردازشی قرار می‌دهیم. بر اساس اولویت تعیین‌شده، هر بار یالی را که دارای بیشترین اولویت است انتخاب کرده و در سیکل جاری پردازنده زمان‌بندی می‌کنیم و کاشی مربوطه و پردازنده مربوطه را به همراه یال‌های متصل به آن‌ها حذف می‌کنیم. درصورتی‌که پردازنده‌ها و همچنین کاشی‌هایی باقی بمانند که دارای ارتباط نیستند (وزن یال‌های مربوطه صفر در نظر گرفته می‌شود) یعنی هیچ اشتراک داده‌ای وجود نداشته باشد، وزن تمامی یال‌ها برابر با میزان اشتراک داده‌ای هر کاشی با کاشی‌های درون حافظه نهان سطح بعدی پردازنده (که در لیست حافظه‌های نهان پردازنده مشخص شده است) انتخاب می‌شود. این عمل تا زمانی که سیکل جاری تمام هسته‌های پردازشی پر شود و یا لیست کاشی‌های آزاد تهی شود ادامه می‌یابد. سپس به‌منظور زمان‌بندی سیکل بعدی، تمام کاشی‌های وابسته به کاشی‌های زمان‌بندی‌شده را به لیست کاشی‌های آزاد اضافه کرده و این فرایند مجدداً تکرار می‌شود.

به‌عنوان‌مثال، گراف اشتراک داده‌ای نشان داده‌شده در شکل (۱) را در نظر بگیرید. در سیکل اول جدول زمان‌بندی، کاشی‌های ۰ و ۱ به هسته‌های پردازشی ۱ و ۲ تخصیص داده می‌شود. برای زمان‌بندی سیکل بعدی هسته‌های پردازشی، یک گراف دوطرفه بین سایر کاشی‌ها و دو هسته پردازشی ایجاد می‌شود. وزن یال‌ها را نیز برابر با وزن یال بین هر کاشی و کاشی تخصیص داده‌شده به پردازنده در سیکل قبل در نظر می‌گیریم. با ایجاد این گراف، یال بین کاشی ۲ و پردازنده ۱ و همچنین بین کاشی ۳ و پردازنده ۲ برابر با ۱۰۸۱۰ و بیشینه است. درنتیجه کاشی ۲ به پردازنده ۱ و کاشی ۳ به پردازنده ۲ تخصیص می‌یابد. همین مراحل تا تخصیص تمامی کاشی‌ها به پردازنده‌ها ادامه می‌یابد.

با استفاده از این الگوریتم تا حد امکان دسترسی پردازنده‌ها به داده‌ها از طریق حافظه نهان اختصاصی پردازنده و با سرعت بالاتر انجام می‌شود. همچنین با توجه به در نظر گرفتن سایر سطوح حافظه نهان درصورتی‌که اشتراک داده‌ای با حافظه اختصاصی هسته‌های پردازشی وجود نداشته باشد، هنگامی‌که کاشی‌ها دارای اشتراک RAR هستند دسترسی پردازنده‌های مختلف به داده‌ها در یک سیکل صورت می‌گیرد و بدین ترتیب حداکثر استفاده از داده‌های واردشده به تراشه پردازنده صورت می‌گیرد. بدین ترتیب، الگوریتم سعی در به دست آوردن بخش‌های موازی و همچنین حداکثر کردن استفاده مجدد از داده‌ها و حداقل کردن دسترسی به حافظه خارج از تراشه دارد.

شبه‌کد الگوریتم زمان‌بندی کاشی‌ها در الگوریتم ۱ نشان داده شده است. مفهوم نمادهای به‌کاررفته در الگوریتم به‌این‌ترتیب است. ورودی الگوریتم گراف استفاده مجدد از داده‌ها DRG(V,E) است. V مجموعه گره‌های گراف است که درواقع همان کاشی‌ها هستند. E مجموعه یال‌های گراف است که مشخص‌کننده وابستگی بین دو کاشی است. بر این اساس برای هر گره v عضو V، Data(v) مشخص‌کننده مجموعه داده‌های دسترسی شده در کاشی v است. همچنین برای هر v ، CriticalPath(v) مشخص‌کننده اندازه طولانی‌ترین مسیر از گره v تا یک گره پایانی تعریف می‌شود. هر $e=(v_1,v_2)$ عضو E دارای وزن Weight(e) است که برابر میزان اشتراک داده‌ای بین دو کاشی است و به‌صورت (۵) تعریف می‌شود:

$$Weight(e)=\left|Data(v1)\cap Data(v2)\right| \qquad (۵)$$

نوع هر لبه e بر اساس نوع وابستگی، که می‌تواند RAR یا غیر RAR باشد، به ترتیب RAR و یا Non-RAR قرار داده می‌شود. درنهایت P مشخص‌کننده مجموعه هسته‌های پردازشی در دسترس است و برای هر p عضو P ، cache[p][cacheLevel] مشخص‌کننده مجموعه داده موجود در حافظه نهان سطح cacheLevel از هسته پردازشی p است. در اینجا سطوح حافظه نهان به ترتیب از داخلی‌ترین سطح که حافظه نهان اختصاصی هسته پردازشی است با عدد ۱ مشخص می‌شود.

در شکل (۲) کد کاشی‌بندی شده اولیه مربوط به ضرب ماتریس-ها، گراف استفاده مجدد داده‌ها، ماتریس زمان‌بندی برای دو هسته پردازشی و درنهایت کد ایجادشده برای هر ریسمان[۱۶] مشخص شده است. همان‌طور که مشخص است هر ریسمان ستون مربوط به هسته پردازشی که قرار است بر روی آن اجرا شود را به‌عنوان ورودی دریافت کرده و به ترتیب برای تمام سیکل‌های زمانی هر بار یک کاشی را اجرا می‌کند. درصورتی‌که نیاز به همگام‌سازی بین ریسمان‌ها باشد، عمل همگام‌سازی ریسمان‌ها در سیکل مربوطه انجام می‌شود. با استفاده از الگوریتم زمان‌بندی پیشنهادی، با توجه به اشتراک داده‌ها، کاشی‌ها به گروه‌هایی بخش‌بندی می‌شوند که می‌توانند با کمترین هزینه همگام‌سازی، به‌صورت موازی بر روی پردازنده‌های مختلف اجرا شوند و کاشی‌های درون هر گروه بیشترین اشتراکات داده‌ای را باهم دارند. درصورتی‌که تعداد کاشی‌ها زیاد باشد می‌توان از یک الگوریتم پیش‌زمان‌بندی خوشه‌بندی استفاده کرد.

Algorithm 1. Proposed Tile Scheduling Algorithm
Input: Data Reuse Graph (DRG) (V,E), V is set of tiles, E is set of tile dependencies
Output: Schedule table

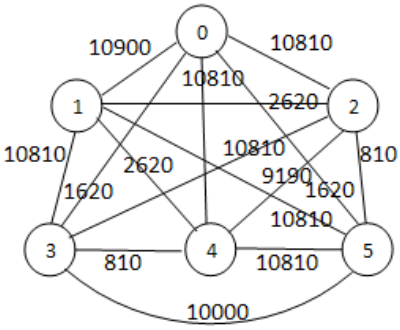
1. *Cycle*=1;
2. P= set of all processor cores; freeNodes= all unscheduled schedulable nodes in DRG
3. NRL= nodes in freeNodes with at least one Non-RAR output edge
4. RL=freeNodes-NRL
5. Select $v \in NRL$ where criticalPath(*v*) is Maximum
6. *Cl*=0;//cacheLevel
7. Select $p \in P$ where weight(U cache[*p*][*cl*],*v*) is Maximum
8. If(MaximumWeight==0)
   Cl++;
   Goto 7;
   Else
      NRL=NRL-*v*;
      P=P-*p*;
      Schedule[*cycle*][*p*]=*v*;
9. While(NRL!=Ø and P!=Ø) goto 5
10. If(P==Ø)
    Cycle++;
    goto2
11. *Cl*=0;
12. Make Two Sided Graph(TSG) between all $v \in RL$ and all $p \in P$
13. Weight(*v*,*p*)=weight(U cache[*p*][*cl*],*v*)
14. Select e=(*v*,*p*), $e \in TSG$ with maximum weight
15. If(e.weight==0)
    Cl++;
    Goto 13;
    Else
    Remove *p* and *v* from TSG;
       RL=RL-*v*;
       P=P-*p*;
       Schedule[*cycle*][*p*]=*v*;
16. While(P!=Ø and RL!=Ø) goto
17. *Cycle* ++
18. While all $v \in V$arenot scheduled goto 1

شکل (۱): شبه‌کد الگوریتم زمان‌بندی کاشی‌ها

درنهایت پس از زمان‌بندی باید کد اجرایی موازی تولید شود. برای این منظور با استفاده از Cloog [۱۶]،که یک ابزار تولید کد از مدل چندوجهی است، با استفاده از تبدیل به‌دست‌آمده در جهت به دست آوردن فضای تکرار قابل کاشی‌بندی، حلقه‌های تکرار قابل کاشی‌بندی تولید می‌شود. سپس با اعمال الگوریتم زمان‌بندی پیشنهادی، بر اساس هسته‌های پردازشی در دسترس، جدول زمان‌بندی و همگام‌سازی استخراج می‌شود. سپس به تعداد هسته‌های پردازشی، ریسمان ایجادشده و کد ایجادشده توسط ابزار Cloog به‌عنوان کد اجرایی ریسمان در نظر گرفته‌شده و همچنین جدول زمان‌بندی مربوطه به‌عنوان ورودی به آن ارسال می‌شود. درنهایت ریسمان‌های مختلف به‌صورت موازی و بر اساس جدول زمان‌بندی مربوطه، کاشی‌های زمان‌بندی‌شده در سیکل‌های مختلف را اجرا می‌کنند و در صورت نیاز، همگام‌سازی بین ریسمان‌های مختلف در سیکل‌های موردنیاز صورت می‌گیرد.

```
int[][] C = new int[M][N];
int[][] A = new int[M][K];
int[][] B = new int[K][N];
lb1 = 0,ub1 = floord(M - 1, tileSize);
lb2 = 0,ub2 = floord(N - 1, tileSize);
lb3 = 0,ub3 = floord(K - 1, tileSize);
for (t1 = lb1; t1 <= ub1; t1++)
  for (t2 = lb2; t2 <= ub2; t2++)
    for (t3 = lb3; t3 <= ub3; t3++)
      for (t4 = max(0, tileSize * t1); t4 <= min(M - 1,tileSize * t1 + tileSize-1);t4++)
        for (t5 = max(0, tileSize * t2); t5 <= min(N - 1,tileSize * t2 + tileSize-1);t5++)
          for (t6 = max(0, tileSize * t3); t6 <= min(K - 1,tileSize * t3 + tileSize-1);t6++)
              C[t4][t5] = C[t4][t5] + A[t4][t6]* B[t6][t5];
```

(a) Tiled code



Schedule table

Cycle	Core 1(t1,t2,t3)	Core 2(t1,t2,t3)
1	0=(0,0,0)	1=(0,0,1)
2	2=(0,1,0)	3=(0,1,1)
3	4=(0,2,0)	5=(0,2,1)

(b) Data reuse graph and virtual scheduling table

```
Thread m:
Do for each (t1,t2,t3) in schedule[i][Core m]

  for (t4 = max(0, tileSize * t1); t4 <= min(M - 1,tileSize * t1 + tileSize-1);t4++)
    for (t5 = max(0, tileSize * t2); t5 <= min(N - 1,tileSize * t2 + tileSize-1);t5++)
      for (t6 = max(0, tileSize * t3); t6 <= min(K - 1,tileSize * t3 + tileSize-1);t6++)
          C[t4][t5] = C[t4][t5] + A[t4][t6]* B[t6][t5];
```

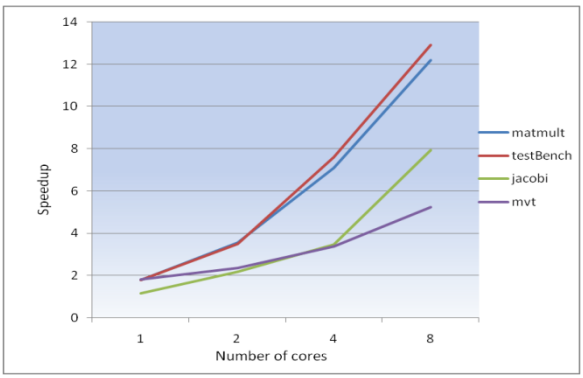
(c) Scheduled tiled code

شکل (۲): مثالی از کاشی‌بندی و زمان‌بندی کاشی‌ها؛ (a) حلقه تودرتوی محاسبه ضرب ماتریس‌ها، (b) گراف استفاده مجدد داده‌ها بین کاشی‌ها و جدول زمان‌بندی کاشی‌ها، (c) کد نهایی کاشی‌بندی و زمان‌بندی‌شده برای یک ریسمان

۵- نتایج تجربی

به‌منظور پیاده‌سازی و ارزیابی روش پیشنهادی از Clan برای به دست آوردن مدل چندوجهی برنامه و از Candl برای به دست آوردن چندوجهی وابستگی استفاده شده است [۱۷]. همچنین از Cloog به‌منظور تولید کد تبدیل یافته استفاده شده است. برای ارزیابی کارایی روش پیشنهادی میزان افزایش کارایی برنامه‌های بهینه‌شده نسبت به حالت ترتیبی برنامه‌ها بر روی برنامه‌های محک مختلف شامل محاسبات ژاکوبی، ضرب ماتریسی، mvt و یک برنامه محک به نام testBench شامل سه حلقه تودرتو بدون وابستگی غیرRAR و حاوی درصد بالایی از استفاده مجدد داده‌ها که به‌عنوان یک نمونه کد ساده با قابلیت بهبود محلیت و استخراج توازی بالا در نظر گرفته شده است. توجه شود که به‌منظور ایجاد فضای تکرار قابل کاشی‌بندی، از رویکرد ارائه‌شده در [۹] استفاده شده است.

در این مقاله نشان داده شده که چگونه با ارائه ترکیب جدیدی از راهکارهای موازی‌سازی حلقه‌ها و بهبود محلیت داده‌ها برای دسترسی‌های انجام‌شده در حلقه‌های تودرتو سرعت اجرایی ضرب ماتریسی بر روی ۸ پردازنده حدود ۱۲ برابر نسبت به کد غیربهینه اولیه افزایش داده شده است. شناخته‌شده‌ترین ابزار در این زمینه پولوتو است. با ابزار پولوتو با شرایطی مشابه سرعت الگوریتم ضرب حدود ۱۱ برابر شده است. علت اصلی بهبود عملکرد، زمان‌بندی کاشی‌ها با در نظر گرفتن سلسله‌مراتب حافظه در پردازنده بوده است. توجه شود که بر طبق قانون مور با در دست داشتن ۸ پردازنده حداکثر می‌توان سرعت اجرایی یک برنامه را ۸ برابر کرد که در روش پیشنهادی افزایش سرعت ضرب ماتریسی نسبت به حالت بهینه‌شده ترتیبی حدود ۶،۳ است.



شکل (۳): افزایش سرعت به‌دست‌آمده برای تعداد هسته‌های پردازشی مختلف

شکل (۴): میانگین کاهش فقدان داده در سطوح L1 و L2 حافظه نهان

شکل (۳) میزان افزایش سرعت اجرایی برنامه‌های بهینه‌شده با روش پیشنهادی را در مقایسه با کد اصلی برنامه‌ها بر روی یک کامپیوتر با مشخصات Intel Xeon E5620 با ۸ هسته پردازشی، ۱۲ مگابایت حافظه نهان اشتراکی، ۳۲ کیلوبایت حافظه نهان اختصاصی برای هر هسته و ۴۸ گیگابایت حافظه اصلی نشان می‌دهد. همان‌طور که مشخص است روش پیشنهادی با در نظر گرفتن موازی‌سازی و بهبود محلیت به‌عنوان فاکتورهای اصلی در بهینه‌سازی برنامه‌ها سرعت اجرایی برنامه‌ها را افزایش داده و در بعضی موارد این افزایش سرعت بیشتر از نسبت خطی تعداد هسته‌های پردازشی است. این عملکرد، حاصل در نظر گرفتن بهبود محلیت داده‌ها در کنار بخش‌بندی مناسب داده‌ها و حلقه‌ها برای اجرا بر روی چندپردازنده‌ها است.

شکل (۴) میزان کاهش فقدان داده‌ها[۱۷] در حافظه نهان اختصاصی (L1) و همچنین حافظه نهان اشتراکی (L2) بین هسته‌های پردازشی مختلف را نشان می‌دهد. این نتایج با اجرای برنامه‌ها بر روی یک چندپردازنده شبیه‌سازی‌شده با مشخصات ۴ هسته پردازشی، ۱ مگابایت حافظه نهان اشتراکی، ۶۴ کیلوبایت حافظه اختصاصی با الگوریتم جایگذاری حافظه نهان LRU به دست آمده است. همان‌طور که مشخص است، الگوریتم پیشنهادی با کاشی‌بندی فضای تکرار و زمان‌بندی آگاه از سلسله‌مراتب حافظه نهان با ارضای درصد بالایی از استفاده مجدد داده‌ها در حافظه نهان اختصاصی و اشتراکی درون تراشه منجر به کاهش دسترسی به حافظه خارج از تراشه شده و درنتیجه بهبود کارایی برنامه می‌شود.

درنهایت بهبود در زمان اجرایی برنامه‌های محک برای ۸ هسته پردازشی در مقایسه با پلوتو به‌طور میانگین ٪۱۱ بوده است. همچنین زمان کامپایل برنامه‌ها بسیار ناچیز بوده و قابل توجه نیست.

۶- نتیجه

هدف اصلی این مقاله موازی‌سازی و بهبود محلیت داده‌ها برای حلقه‌های تودرتو است به نحوی که توازی دانه‌درشت با در نظر گرفتن استفاده مجدد داده‌ها برای اجرا بر روی پردازنده‌های چندهسته‌ای حاصل شود. برای رسیدن به این هدف از تبدیل کاشی‌بندی فضای تکرار حلقه‌ها استفاده شده است. در روش پیشنهادی شکل و اندازه کاشی‌ها بر اساس سلسله مراتب حافظه نهان و با در نظر گرفتن معیار بهبود محلیت داده‌ها تعیین می‌شود. سپس زمان‌بندی کاشی‌ها برای اجرا بر روی چندپردازنده‌ها با در نظر گرفتن میزان اشتراکات داده‌ای بین کاشی‌ها صورت گرفته و کد اجرایی موازی، بر اساس جدول زمان‌بندی ایجاد شده در زمان کامپایل، تولید می‌شود. با توجه به اینکه در روش پیشنهادی موازی‌سازی حلقه‌ها در کنار بهبود محلیت قرار گرفته است، بخش‌های موازی ایجاد شده دارای محلیت دسترسی به داده‌ها بوده و همچنین داده‌های اشتراکی بین بخش‌ها تا حد امکان در سیکل‌های زمانی نزدیک به هم زمان‌بندی می‌شوند و منجر به افزایش کارایی برنامه‌ها می‌شود. نتایج ارزیابی روش پیشنهادی بر روی برنامه‌های مختلف نشان از کارایی بالای روش پیشنهادی در بهبود کارایی برنامه‌ها دارد.

مراجع

[1] G. Ottoni, *Global Instruction Scheduling for Multi-Threaded Architectures*, PhD Thesis, Princeton University, 2008.

[2] O. Ozturk, "Data locality and parallelism optimization using a constraint-based approach," Journal of Parallel and Distributed Computing, vol. 71, no. 2 , pp. 280-287, 2011.

[3] U. Bondhugula, A. Hartono, J. Ramanujam and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," ACM SIGPLAN Notices, vol. 43, no. 6, pp. 101-113, 2008.

[4] S. Lotfi and S. Parsa, "Parallel loop generation and scheduling," The Journal of Supercomputing, vol. 50, no. 3, pp. 289-306, 2009.

[5] M. E. Wolf and M. S. Lam, "A loop transformation theory and an algorithm to maximize parallelism," IEEE Transactions on Parallel and Distributed Systems, vol. 2, no. 4, pp. 452-471, 1991.

[6] J. Xue and C-H. Huang, "Reuse-driven tiling for improving data locality," International Journal of Parallel Programming, vol. 26, no. 6, pp. 671-696, 1998.

[7] Y. Song and Z. Li, "New tiling techniques to improve cache temporal locality," ACM SIGPLAN Notices, vol. 34, no. 5, pp. 215-228, 1999.

[8] M. E. Wolf and M. S. Lam, "A data locality optimizing algorithm", ACM SIGPLAN Notices, vol. 26, no. 6, pp. 30-44, 1991.

[9] S. Parsa and M. Hamzei, "Locality conscious nested-loops parallelization," ETRI Journal, vol. 36, no. 1, pp. 124-133, 2014.

[10] J. Liu, Y. Zhang, W. Ding and M. Kandemir, "On-chip cache hierarchy-aware tile scheduling for multicore machines," In 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp. 161-170. 2011.

[11] L. Pouchet, *Iterative Optimization in the Polyhedral Model*, PhD Thesis, France University of Paris-Sud XI, 2010.

[12] A. Cohen, S. Girbal and O. Temam, "A polyhedral approach to ease the composition of program transformations," In Euro-Par 2004 Parallel Processing, pp. 292-303, 2004.

[13] L. Pouchet, C. Bastoul, A. Cohen and J. Cavazos, "Iterative optimization in the polyhedral model: part II, multidimensional time", ACM SIGPLAN Notices, vol. 43, no. 6, pp. 90-100, 2008.

[14] P. Feautrier, "Some efficient solutions to the affine scheduling problem. part II. multidimensional time," International Journal of Parallel Programming, vol. 21, no. 6, pp. 389-420, 1992.

[15] J. Ramanujam and P. Sadayappan, "Tiling multidimensional iteration spaces for multicomputers," Journal of Parallel and Distributed Computing, vol. 16, no. 2, pp. 108-120, 1992.

[16] C. Bastoul, "Efficient code generation for automatic parallelization and optimization," In ISPDC'2 IEEE International Symposium on Parallel and Distributed Computing, pp. 23-30, 2003.

[17] C. Bastoul, "Extracting polyhedral representation from high level languages," Technical Report at Paris-Sud University, 2008.

زیرنویس‌ها

[1] Multi-core processor
[2] Cache
[3] Data reuse
[4] Data Locality
[5] Iteration space tiling
[6] Wavefront
[7] Polyhedral model
[8] Private cache
[9] Shared cache
[10] Unimodular transformation
[11] Pluto
[12] Uniform dependency
[13] Clustering algorithm
[14] Coarse grain
[15] Fully permutable
[16] Thread
[17] Cache miss