مجلة جامعة طرطوس للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (4) العدد(6) 2020
Tartous University Journal for Research and Scientific Studies - engineering Sciences Series Vol. (4) No. (6) 2020

تقسيم البيانات بين ذاكرة SPM والذاكرة الرئيسية في نظام مضمن

م. رزان عقباتي*

(تاريخ الإيداع 7/20 /2020. قُبِل للنشر في 2020/9/13)

□ ملخّص □

مع ازدياد أهمية استخدام النظم المضمنة في يومنا هذا، اتجهت البحوث العلمية إلى هذه النظم بشكل كبير، بحيث يسعى مصممو هذا النوع من النظم باستمرار حتى يومنا هذا إلى تحسين الأداء وتقليل استهلاك الطاقة.

وبما أن الذواكر تُعَد من أهم الأجزاء التي تستهلك طاقة في المعالجات الحديثة، كما أننا نهتم بها من ناحية سرعة النظام، بحيث أتاحت لنا هرمية الذاكرة اختصاراً كبيراً للزمن، بات الاستخدام الأمثل للذواكر في الأنظمة المضمنة مجالاً مهماً للبحث وللدراسة.

في بحثنا هذا سوف نتعرف على ذاكرة Scratch_pad (SPM) التي يتم استخدامها بشكل واسع في معظم تطبيقات النظم المضمنة لما تتمتع به من مزايا في هذا المجال، وسندرس العوامل المؤثرة في تقسيم البيانات لأي تطبيق بين ذاكرة SPM والذاكرة الرئيسية التي تتوضع خارج الشريحة والتي يتم الوصول إليها عن طريق الذاكرة المخبأة، لنصل في نهاية بحثنا إلى خوارزمية عامة لتقسيم البيانات (ثوابت عددية ومصفوفات) بين الذاكرتين السابقتين، وذلك بهدف تخفيض زمن التنفيذ للتطبيقات المضمنة.

الكلمات المفتاحية: الأنظمة المضمنة، ذاكرة SPM، الذاكرة الرئيسية.

*مهندسة في قسم النظم الحاسوبية والالكترونية _ كلية هندسة تكنولوجيا المعلومات والاتصالات _ جامعة طرطوس_ سوريا.

مجلة جامعة طرطوس للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (4) العدد(6) 2020
Tartous University Journal for Research and Scientific Studies - engineering Sciences Series Vol. (4) No. (6) 2020

Partition data between SPM and main memory in an embedded system

Eng. Razan Akabati*

(Received 20/7/2020. Accepted 13/ 9/2020)

□ ABSTRACT □

With the increasing importance of the use of embedded systems , scientific research has turned to these systems significantly, so that designers of this type of system continuously to improve performance and reduce energy.

And since memory is one of the most important parts that consume energy in modern processors, and we also care about it in terms of system speed, so that the memory hierarchy has given us a great abbreviation of time, the optimal use of memory in the embedded systems has become an important area for research and study.

In our research, we will learn about the Scratch_pad (SPM) memory that is widely used in most of the embedded systems applications because of its advantages in this field, and we will study the factors affecting the data partition of any application between the SPM memory and the main memory that is placed outside the chip and that is accessed via cache, let us reach at the end of our research a general algorithm for partition data (numerical constants and matrices) between the previous two memories, with the aim of reducing the execution time of the embedded applications.

Keywords: Embedded systems, SPM memory, main memory.

*Engineer in the Department of Computer and Electronic Systems Engineering _ Faculty of Information and Communications Technology Engineering_ Tartous University_ Syria.

1- المقدمة:

مع ازدياد أهمية النظم المضمنة بشكل كبير، باتت تطبيقات هذه النظم أحد أهم المجالات في يومنا الحالي، ويشكل استهلاك الطاقة عاملاً أساسياً في هذه النظم، فكان لابد من استخدام بعد التقنيات لتقليل استهلاك الطاقة في المعالج، أحد هذه التحسينات المتاحة لمصممي النظام هو اختيار التسلسل الهرمي للذاكرة. وتؤثر القرارات التي تُتخَذ أثناء تصميم الأجهزة والبرمجيات بشكل كبير على أداء نظام الذاكرة، والذي يُعَد مساهماً رئيسياً في إجمالي استهلاك الطاقة وسرعة تنفيذ النظام. حيث أنّ النفاذ إلى الذاكرة الرئيسية كذلك يستهلك كميات كبيرة من الطاقة، ومن الجدير بالذكر أن الذاكرة المخبأة هي واحدة من أعلى المكونات استهلاكاً للطاقة في المعالجات الحديثة. يُذكَر أن الذاكرة المخبأة الخاصة بالتعليمات وحدها تستهلك ما يصل إلى 27٪ من طاقة المعالج [1].

في الآونة الأخيرة ، تركز الاهتمام على وجود ذاكرة SPM لتقليل الطاقة وتحسين الأداء[2]. من ناحية أخرى، يمكن أن تحل هذه الذاكرة محل الذاكرة المخبأة فقط إذا كانت مدعومة من قبل مترجم فعال. الذاكرة المخبأة للتعليمات والبيانات هي الذاكرة المحلية السريعة، وهي بمثابة وسيط بين المعالج والذاكرة خارج رقاقة. بينما ذاكرة Scratch Pad, فهي عبارة عن ذاكرة بيانات صغيرة عالية السرعة يتم تعيين عناوينها إلى مساحة عنوان منفصل عن عناوين الذاكرة الموجودة خارج الشريحة، ولكن متصلة بنفس خطوط العناوين و البيانات. تملك كل من الذاكرة المخبأة وذاكرة _SPM SRAM زمن وصول يساوي دورة وحيدة للمعالج، في حين أن الوصول إلى الذاكرة خارج الشريحة (عادةً DRAM) يأخذ عدة دورات (عادةً ما تكون 10−20) دورة للمعالج [3].

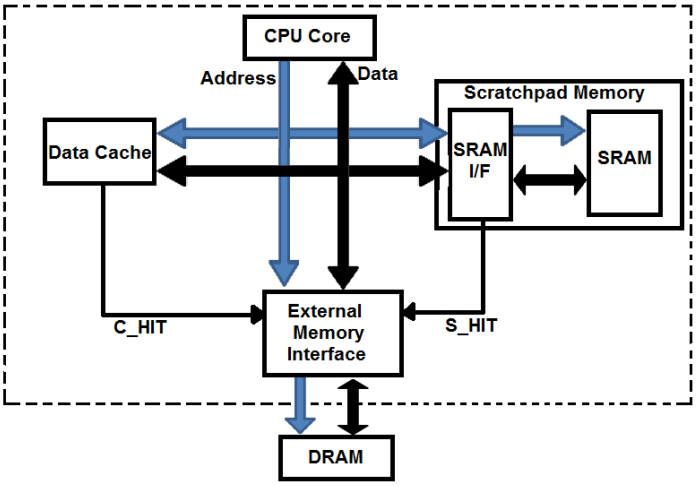
عندما يتم ترجمة (compiling) التطبيق المضمن ، يمكن الآن تخزين البيانات التي سيتم الوصول إليها إما في ذاكرة Scratch−Pad أو في الذاكرة خارج الشريحة. في الحالة الثانية، يتم الوصول إليه بواسطة المعالج من خلال الذاكرة المخبأة للبيانات.

2- أهمية البحث وأهدافه:

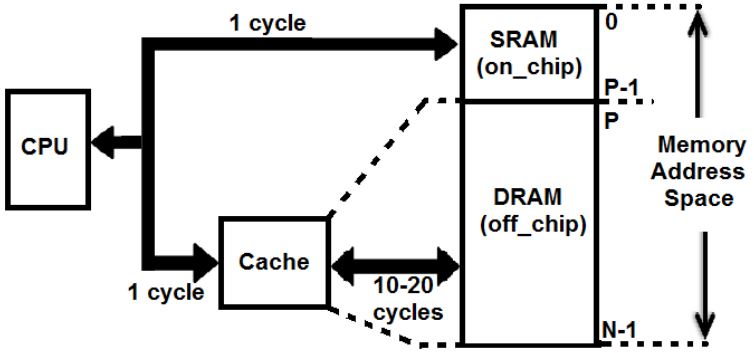
يبين الشكل (1) المخطط الصندوقي لبنية معالج أساسي مضمن نموذجي (على سبيل المثال ، معالج LSI Logic CW33000 RISC [4])، حيث أن الأجزاء الموجودة في المستطيل المنقط تدمج على رقاقة واحدة، والتي تتفاعل مع ذاكرة تقع خارج الرقاقة، وعادة ما يتم تصميم هذه الذاكرة الخارجية باستخدام دارات DRAM. تتصل خطوط العناوين والبيانات من وحدة المعالجة المركزية إلى الذاكرة المخبأة وذاكرة Scratch−Pad و منافذ الذاكرة الخارجية EMI (External Memory Interface). عند الحاجة إلى الولوج من وحدة المعالجة المركزية (CPU) إلى الذاكرة، تظهر الذاكرة المخبأة للبيانات عملية Cache hit إلى كتلة EMI من خلال إشارة C_HIT (وجود البيانات ضمن الذاكرة المخبأة). وبالمثل، إذا حددت دارة منفذ SRAM في ذاكرة Scratch−Pad أن عنوان الذاكرة المشار إليه يعيّن في ذاكرة SRAM على الرقاقة ، فإنه يفترض التحكم في ناقل البيانات ويشير في هذه الحالة إلى EMI من خلال الإشارةS_HIT (وجود البيانات ضمن ذاكرة SPM). في حالة فقد كل من الذاكرة المخبأة وعدم وجود البيانات في ذاكرة SPM_SRAM ، يقوم EMI بنقل كتلة من البيانات بالحجم المناسب (يساوي حجم خط الذاكرة المخبأة بين DRAM والذاكرة المخبأة.

يظهر مخطط فضاء عناوين البيانات في الشكل (2) [3] ، تشغل ذاكرة SPM عناوين الذاكرة من 0 إلى P− 1، وتحتاج إلى دورة واحدة للوصول إلى المعالج. وهكذا ، في الشكل (1) ، سيتم تأكيد S_HIT عندما يحاول المعالج الوصول إلى أي عنوان في النطاق 0 ..P−1. بينما عناوين الذاكرة منP إلى N−1 يتم تخصيصها للذاكرة خارج

الرقاقة (DRAM) ، ويتم الوصول إليها من قبل وحدة المعالجة المركزية من خلال الذاكرة المخبأة للبيانات. ينتج عن ذاكرة المخبأة للعنوان في النطاق P ... 1 –N تأخير دورة واحدة ، في حين يؤدي الفقد في الذاكرة المخبأة ، والتي تؤدي إلى نقل كتلة بين الذاكرة خارج الذاكرة والذاكرة المخبأة ، إلى تأخير 10 –20 دورة .



الشكل (1) المخطط الصندوقي لبنية معالج أساسي مضمن نموذجي [4]



الشكل (2) فضاء عناوين البيانات

سيتم تقديم استراتيجية لتقسيم المتغيرات في شيفرة التطبيق إلى ذاكرة Scratch–Pad و DRAM خارج الشريحة التي يتم الوصول إليها من خلال الذاكرة المخبأة للبيانات، لتحقيق الأداء الأفضل من خلال التوضع الانتقائي في SRAM لتلك المتغيرات التي من المقدر أن تسبب الحد الأقصى لعدد التعارضات في الذاكرة المخبأة للبيانات.

3– الدراسات المرجعية

تلقت ذاكرة SPM اهتماماً كبيراً في النظم المضمنة ونظم الزمن الحقيقي كبديل للذاكرة المخبأة للمعالج. على عكس الذاكرة المخبأة تدعم ذاكرة scratchpad أوقات الوصول المتسقة، والتي يمكن التنبؤ بها، لذلك فإن أي مهمة تنفذ من SPM مضمونة أن يكون لها وقت تنفيذ يمكن التنبؤ به. كما أن SPM أفضل من الذاكرة من حيث المساحة وكفاءة الطاقة [5].

بالاضافة إلى أنه في مجال النظم المضمنة[6]، كانت هناك اقتراحات لبنية ذاكرة متخصصة تجمع مزايا الذاكرة المخبأة وذاكرة SPM. على سبيل المثال تم اقتراح ذاكرة سُميَت ذاكرة TickPad (TPM) [7]. تملك ذاكرة TPM آلية تحميل أجهزة ديناميكية يتم التحكم فيها بشكل ثابت .

وهي الذاكرة الأنسب للغات البرمجة المتزامنة مثل [8] Pret–C التي تأخذ عينات من المدخلات في أوقات منفصلة من الزمن، وتوفر مخرجات في لحظات معروفة باسم وقت التجزئة(tick time). تهدف TPM بشكل عام إلى أن تكون أكثر قابلية للتنبؤ من الذاكرة المخبأة، وأسهل في إدارتها من SPM. ومع ذلك ، يعتمد TPM على لغة برمجة متزامنة ومتخصصة، ويقتصر استخدامها على تطبيق واحد.

كما كان هناك العديد من الأبحاث التي استخدمت تقسيم الذاكرة المخبأة باستخدام العتاد الصلب (hardware) أو في العتاد اللين (software). يعد تلوين الصفحة [9 ، 10] أكثر تقنيات تقسيم الذاكرة المخبأة المعتمدة على العتاد اللين شيوعاً.

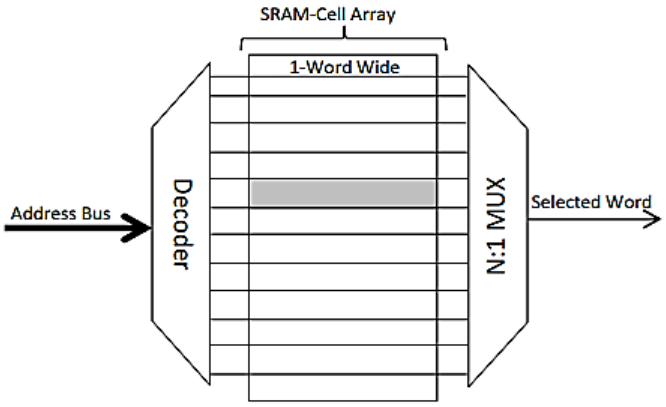
وفي بداية العام 2018 قام الباحث سعود واصلي بدراسة الذاكرة SPM وطرح آلية استخدام هذه الذاكرة في المعالجات متعددة النوى، ولكن الدراسة كانت تشمل دراسة خوارزميات الجدولة المطبقة في نظام التشغيل[11].

4- ذاكرة Scratch_pad

تُعرف ذاكرة SPM (Scratchpad memory) بأنها الذاكرة الداخلية العالية السرعة صغيرة الحجم التي تخزن بيانات صغيرة الحجم والتي ليس من الضروري أن يتواجد نسخة منها في الذاكرة الرئيسية. تمتاز بسرعة الوصول إليها نتيجة قربها من وحدة المعالجة المركزية فإن تأخير الوصول إليها لايذكر بالمقارنة مع الذاكرة الرئيسية.

يظهر الشكل (3) بنية ذاكرة SPM [12] بحيث نلاحظ أنها عبارة عن دارة رقمية بسيطة، وبالتالي تحتل مساحة أصغر وتستهلك طاقة أقل ولا يتطلب الوصول إلى كلمة الذاكرة سوى مفكك تشفير Decoder وناخب Multiplexer.

يسمى الوصول إلى الذاكرة واختيار كلمة معينة بالفهرسة indexing. أثناء الوصول، يتم تحديد خلايا SRAM المستهدفة ثم قراءتها أو كتابتها.



الشكل (3) بنية ذاكرة SPM

الافتراض هو أن ذاكرة SPM تحتل جزءاً واحداً متميزاً من مساحة عناوين الذاكرة مع باقي المساحة التي تشغلها الذاكرة الرئيسية .وبالتالي ، ليست هناك حاجة للتحقق من توافر البيانات / التعليمات في SPM. وهذا يقلل من عمليات المقارنة وتلقى إشارة miss/hit.هذا يساهم في تقليل الطاقة وكذلك خفض المساحة.

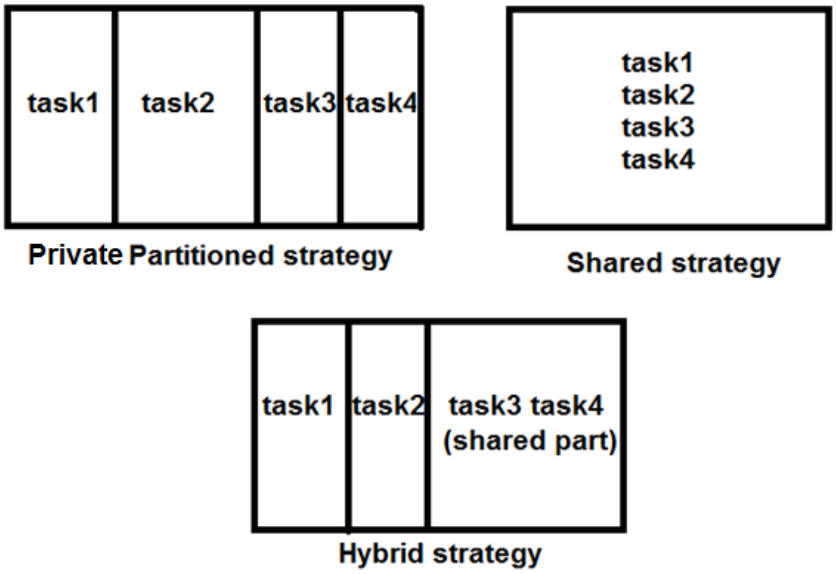
ومن الجدير بالذكر أنه يتم قيادتها والتحكم بها برمجياً (software) وتكون غير مرئية بالنسبة للعتاد الصلب (hardware) [13]، فيجب أن تتوفر لدينا معرفة كبيرة بالمنصة والنظام الذي نتعامل معه، وذلك لكتابة البرنامج الذي

يتحكم بهذه الذاكرة، فيتم التحكم بعملية نقل المهام من الذاكرة الرئيسية إلى SPM والعكس بالعكس عن طريق الجدولة والتي تختلف من نظام إلى آخر، بحيث يكون لدينا برمجية خاصة لكل تطبيق مراد استخدام ذاكرة SPM فيه.

5- تقسيم ذاكرة SPM:

هناك ثلاث استراتيجيات لتعيين المهام إلى ذاكرة scratchpad [13]. وتظهر الاستراتيجيات في الشكل (4).

- التقسيم الخاص: في هذا السيناريو ، يتم تقسيم scratchpad بالكامل إلى n منطقة ويتم تخصيص كل مهمة لمنطقة واحدة. تبقى كل مهمة في القسم المخصص لها حتى يتم إنهاء التطبيق أو مجموعة المهام بأكملها. وبالتالي يكون القسم مخصص لمهمة معينة. لا يجب أن تكون الأقسام ذات أحجام متساوية. يتم إعطاء حجم الذاكرة الإجمالي في هذه الاستراتيجية من خلال جمع أحجام الأقسام الفردية.
- التقسيم المشترك: هنا تشترك جميع المهام في scratchpad. وهنا يجب تحميل محتويات scratchpad وحفظها و / أو استعادتها في كل مفتاح تبديل. يتم أخذ تكلفة كل تبديل في الاعتبار في الخوارزمية. حجم الذاكرة الإجمالي لهذه الاستراتيجية هو حجم الذاكرة المطلوبة من قبل المهمة الأكثر تطلباً للذاكرة.
- التقسيم الهجين: ينتج الأسلوب الهجين من خليط من الاستراتيجيتين السابقتين، حيث يتم تخصيص بعض المهام لمنطقة مقسمة والبعض الآخر إلى منطقة مشتركة. حجم الذاكرة الإجمالي في هذه الاستراتيجية هو مجموع أحجام الذاكرة المطلوبة من قِبل القسم المشترك وأحجام الأقسام الخاصة.



الشكل (4) طرق تقسيم ذاكرة SPM

6- استراتيجية التقسيم:

الهدف الأساسي لتقسيم متغيرات البرنامج إلى ذاكرة Scratch-Pad و DRAM هو السعي لتقليل التداخل المتبادل بين المتغيرات المختلفة في الذاكرة المخبأة للبيانات. أولاً سنحدد العوامل التي تؤثر على التقسيم، ومن ثم تقديم استراتيجية تقسيم استناداً إلى هذه العوامل.

6-1 العوامل التي تؤثر على عملية التقسيم:

6-1-1 المتغيرات والثوابت العددية:

نقوم بوضع كل المتغيرات والثوابت العددية إلى ذاكرة SPM، وذلك لمنع التداخل مع المصفوفات في الذاكرة المخبأة الخاصة بالبيانات. إذا تم توضع الأعداد إلى DRAM فمن المستحيل تجنب التعارض في مصفوفات الذاكرة المخبأة، لأنه يتم تعيين المصفوفات إلى كتل متجاورة من الذاكرة، أجزاء من هذه المصفوفات سيتم تعيينها في نفس السطر من الذاكرة المخبأة كأعداد، مما تسبب في مشكلة فقد [15].

يستند قرار وضع جميع الأعداد إلى ذاكرة (SPM_SRAM) إلى ملاحظتنا أنه بالنسبة لمعظم التطبيقات، فإن المساحة الذاكرية المحجوزة إلى الأعداد لا تكاد تذكر مقارنة بالمستويات التي تشغلها المصفوفات.

6-1-2 حجم المصفوفة:

توضع المصفوفات التي يكون حجمها أكبر من حجم ذاكرة SPM_SRAM في الذاكرة الخارجية off_chip، بحيث يتم الوصول إلى هذه المصفوفات من خلال الذاكرة المخبأة للبيانات. يتم وضع المصفوفات كبيرة إلى الذاكرة المخبأة هو الخيار الطبيعي، لأنه يبسط عنونة المصفوفة. إذا كان جزء من المصفوفة يتم تعيينه إلى SRAM، فسيتعين على المترجم أن يولد رمزاً يتتبع أي منطقة من المصفوفة يتم التعامل معه، مما يجعل الشفرة البرمجية معقدة وغير فعالة. بالإضافة إلى ذلك، نظراً لأن معظم الحلقات تصل إلى عناصر المصفوفة بشكل منتظم في معظم الأوقات، فمن غير المجدي تعيين أجزاء مختلفة من نفس المصفوفة في ذواكر ذات الخصائص المختلفة.

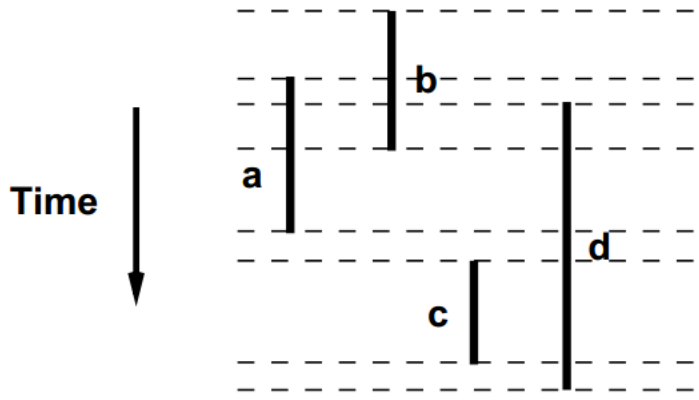
6-1-3 زمن حياة المتغير

يعتبر زمن حياة المتغير، والذي يتم تعريفه بأنه الفترة بين تعريف هذا المتغير والاستخدام الأخير له [15] ، مقياساً مهماً يؤثر على تخصيص الذاكرة. يمكن تخزين المتغيرات التي تملك فترات حياة منفصلة في نفس الموقع، والمصفوفات المختلفة يتم تخصيصها في نفس فضاء الذاكرة.

يمكن أيضاً استخدام معلومات فترة الحياة لتجنب التعارضات المحتملة بين المصفوفات.

لتبسيط تحليل التعارض يوضح الشكل (5) مثال عن توزيع زمن الحياة لأربع متغيرات a,b,c,d. لدينا b و c يملكان زمني حياة منفصلين ، لذلك يمكن إعادة استخدام مساحة الذاكرة المخصصة للمصفوفة b لتستخدم للمصفوفة c. بينما المصفوفتين a و d لهما زمن حياة متداخل، فيمكن تجنب تعارضات الذاكرة المخبأة فيما بينهما عن طريق تعيين أحدهما إلى SRAM.

ومع ذلك ، نظرًا لأن الموارد على الشريحة (مساحة SPM_SRAM) تكون قليلة ، نحتاج إلى تحديد المصفوفة الأكثر أهمية. ويتوقف التحديد لهذه الأهمية الحرجة على تواتر الوصول إلى المتغيرات.



الشكل (5) مثال عن توزيع زمن الحياة لأربع متغيرات

6-1-4 Access Frequency of Array Variables تواتر الوصول لمتغيرات المصفوفة:

للحصول على تفصيل أكثر دقة عن مدى التعارضات في الذواكر، يجب علينا دراسة تواتر الوصول.

على سبيل المثال، في الشكل (5) نلاحظ أن المصفوفة d تملك احتمالية كبيرة بأن تسبب اعتراضات مع المصفوفات الثلاثة الأخرى (a,b,c)، لذلك نقرر تخزين d في ذاكرة SPM_SRAM. لكن إذا كان عدد الوصولات إلى d صغير نسبياً، فمن الأفضل تخزين المصفوفات الأخرى ذات عدد الوصولات الأكبر في SPM، في هذه الحالة لايلعب وجود d في الذاكرة المخبأة دوراً أساسياً في حدوث الاعتراضات.

ومن أجل كل متغير u نعرف بارامتر VAC(u) (variable access count) وذلك لتحديد عدد الوصولات إلى المتغير خلال فترة حياته.

وبشكل مشابه لذلك، عدد الوصولات إلى المتغيرات الأخرى خلال فترة حياة متغير ما هو مؤشر مهم على حدوث التعارضات في الذاكرة المخبأة. فمن أجل كل متغير u، نعرف بارامتر IAC(u) interference access count لتحديد عدد الوصولات للمتغيرات الأخرى خلال فترة حياة u.

نلاحظ أن كل من العوامل التي نوقشت أعلاه، VAC(u) و IAC(u)، إذا تم أخذها بشكل فردي، يمكن أن تعطي فكرة مضللة حول الصراعات المحتملة التي تتعلق بالمتغير u، فمن الواضح أن الصراعات تتحدد بصورة مشتركة من خلال هذين العاملين معاً.

لذلك مجموع هذين البارامترين يعطي مؤشراً جيداً على الصراعات التي تنطوي على المصفوفة u، لذلك نحدد بارامتر جديد هو عامل التداخل IF (interference factor) للمتغير u، ويعطى بالعلاقة:

$$IF(u) = VAC(u) + IAC(u)$$

تشير قيمة IF العالية للمتغير u إلى أنه من المحتمل أن يكون u متضمناً عدد كبير من التعارضات في الذاكرة المخبأة إذا تم تعيينه إلى DRAM. وبالتالي ، نختار توضع المتغيرات التي تملك قيم IF عالية في SRAM.

6-1-5 التعارضات في الحلقات

في الفقرة 6-1-4، عرفنا بارامتر IF الذي يستخدم لتحديد التعارضات في حال كان النص البرمجي code خالٍ من الحلقات (nonloop code). أما في حالة المصفوفات التي يتم الوصول إليها في الحلقات،

من الممكن إجراء تمييز أكثر دقة استناداً إلى أنماط الوصول إلى المصفوفة. نأخذ بعين الاعتبار مقطع من النص البرمجي لثلاثة مصفوفات a,b,c الوصول إليها كما هو موضح في الشكل (6 a).

نلاحظ أن المصفوفتين a,b لديهما نمط وصول متماثل، وهو يختلف عن ذلك الخاص بالمصفوفة c. يمكن استخدام تقنيات محاذاة البيانات [16] لتجنب التعارضات في الذاكرة المخبأة للبيانات بين a و b فيمكن أن تتوضع المصفوفات بشكل مناسب في الذاكرة بحيث أنها لا تتعارض في الذاكرة المخبأة.

ومع ذلك، عندما تكون أنماط الوصول مختلفة، لا يمكن تجنب التعارضات في الذاكرة المخبأة (على سبيل المثال، b,c أنماط الوصول مختلفة بسبب اختلاف المعاملات لمتغيرات الحلقة).

في مثل هذه الظروف، يمكن الحدّ من التعارضات عن طريق تعيين إحدى المصفوفات المتعارضة إلى SRAM. على سبيل المثال، يمكن إزالة التعارضات في المثال أعلاه عن طريق تعيين a و b إلى DRAM/cache و c إلى ذاكرة Scratch–Pad.

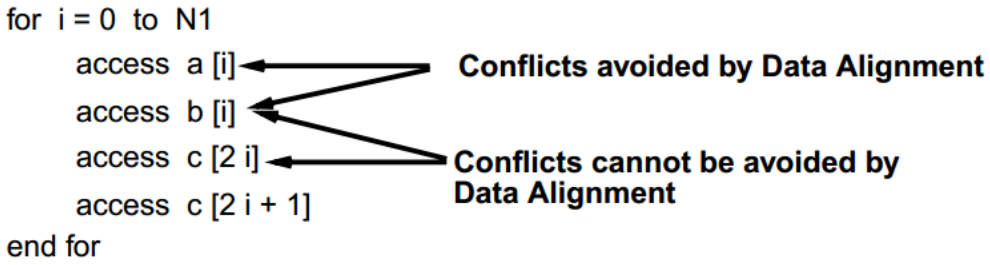
لتحقيق ذلك، نعرف بارامتر عامل صراع الحلقة LCF (loop conflict factor) لمتغير u كما يلي:

$$LCF(u) = \sum_{i=1}^{p}(k(u) + \sum_{v} k(v))$$

حيث المجموع $\sum_{i=1}^{p}$ هو لجميع الحلقات التي يصل إليها المتغير u، و$\sum_{v}$ هو لجميع المصفوفات الأخرى ماعدا u والتي تدخل في الحلقة i، والتي لا يمكنها استخدام تقنيات توضع البيانات للقضاء بشكل كامل على التعارضات في الذاكرة المخبأة مع المتغير u. في المثال أعلاه، حيث لدينا فقط حلقة واحدة (p=1)، فيتم إنشاء قيم LCF كما هو موضح في الشكل 6 (b).

لدينا دخول واحد للمتغير a ودخولين للمتغير c في تكرار واحد من الحلقة ويمكننا إزالة التعارضات في الذاكرة المخبأة للمتغيرين a,b بشكل كامل باستخدام تقنيات توضع البيانات.

قيمة LCF يعطينا مقياس لمقارنة الصراعات الحرجة في حلقة لكافة المصفوفات. بشكل عام، كلما ارتفع رقم LCF، كلما زاد عدد التعارضات للمصفوفة، وبالتالي، كلما كان من المرغوب فيه تعيين المصفوفة إلى ذاكرة Scratch–Pad.



(a)

LCF (a) = k (a) + (k (c)) = N + 2 N = 3N

LCF (b) = k (b) + (k (c)) = N + 2 N = 3N

LCF (c) = k (c) + (k (a) + k(b)) = 2N + N + N = 4 N

(b)

الشكل (6) (a) مثال عن الحلقات (b) حساب قيمة LCF

6-2 مشكلة التقسيم:

في الفقرتين 6-1-4 و 6-1-5 عرفنا بارامترين IF و LCF. وبدمج هذين البارامترين بحيث نصل إلى القيمة التقديرية لعدد الوصولات التي تقودنا إلى التعارضات التي تتسبب بها مصفوفة ما. ونعرف بارامتر جديد هو عامل التعارض الكلي Total Conflict Factor (TCF) لمصفوفة (u) كما يلي:

TCF(u)= IF(u)+ LCF(u)

حيث يتم حساب قيمة IF(u) لفترة الحياة التي نستثني منها المناطق من النص البرمجي حيث يتم إدخال u ضمن حلقة. أما TCF(u) تعطي مؤشراً عن عدد الوصولات الكلية التي تسبب تعارض في الذاكرة المخبأة متضمنة المصفوفة u و لذلك تدل على

أهمية توضع u في ذاكرة SRAM.

سنصيغ مشكلة تقسيم البيانات كما يلي:

لتكن لدينا مجموعة من n مصفوفة $A_1 \ldots\ldots A_n$ ، مع قيم TCF هي $TCF_1 \ldots\ldots TCF_n$ ، وحجمها $S_1 \ldots\ldots S_n$، وحجم ذاكرة SPM_SRAM هو S، والمطلوب إيجاد المجموعة الجزئية المثالية $Q \subseteq \{1,2 \ldots\ldots, n\}$ التي تحقق $\sum_{i\in Q} S_i \leq S$ و $\sum_{i\in Q} TCF_i$ قيمته أعظمية.

6-3 حل مشكلة التقسيم:

في خوارزمية البحث الشامل (exhaustive-search algorithm) لحل مشكلة تعيين الذاكرة يجب أولاً أن تولد مجموعات جزئية تضم كل مجموعة كافة مجموعات المصفوفات المتوافقة (المصفوفات التي يمكن أن تشترك في نفس مساحة SRAM)، ومن ثم بعد ذلك إنشاء كافة المجموعات الممكنة من هذه المجموعات الجزئية، واختيار المجموعة مع الحجم الإجمالي المناسب في SRAM الذي يزيد من القيمة التراكمية لبارامتر TCF. يتطلب هذا الإجراء $O(2^{2^n})$ من الوقت، وهو أمر مكلف بشكل غير مقبول، لأن التابع $y = 2^{2^n}$ يزداد بسرعة كبيرة، حتى بالنسبة للقيم الصغيرة من n.

في حلنا لمشكلة تقسيم بيانات الذاكرة، نقوم أولاً بتجميع المصفوفات التي يمكن أن تشارك مساحة SRAM في مجموعات، ثم استخدام التباين الموجود في خوارزمية تقريب كثافة القيمة (value-density approximation algorithm) [17] لمشكلة تعيين مجموعات إلى SRAM.

تقوم خوارزمية التقريب أولاً بفرز كافة العناصر من حيث القيمة لكل وزن (أي كثافة القيمة)، وتختار العناصر بترتيب متناقص لكثافة القيمة حتى لا يمكن أن تكون هناك عناصر أخرى مُمكنة. نحن نحدد كثافة الوصول (AD) (access density) للكتلة c كما يلي:

$$AD(c) = \frac{\sum_{v \in c} TCF(v)}{\max\{size(v) | v \in c\}}$$

ويستخدم هذا العامل لتعيين الكتل من المصفوفات في SRAM. ونلاحظ أن المقام هو حجم أكبر مصفوفة في الكتلة. وذلك لأنه في أي وقت، واحدة فقط من المصفوفات تكون نشطة، وبالتالي المصفوفات، التي تشترك في نفس فضاء الذاكرة، تحتاج إلى تعيين ذاكرة أكبر في الكتلة فقط.

خوارزمية تقسيم بيانات الذاكرة Memory Assignis تكون كما يلي:

```
Algorithm MemoryAssign
Input: Application program P with Register-allocated variables marked;
  SRAM-Size: Size of Scratch-Pad SRAM
Output: Assignment of arrays to SRAM/DRAM
                                        AvSpace=SRAM-Size -- مساحة ذاكرة SRAM المتاحة
Let U={ array u|u is an array in P} --P هي المجموعة من كل المصفوفات السلوكية في البرنامج
U
Let W=ø -- V هو مجموعة من المصفوفات التي تم إسنادها إلى الذاكرة الخارجية من أجل كل المتغيرات
W
If v is a scalar variable or constant
      Assign v to SRAM
      AvSpace = AvSpace-size(v)
Else
      If size(v)>SRAM_Size
            W=W∪{V} -- DRAM إسناد v إلى
      end if
end if
end for
                                        إنشاء المخطط التوافقي G من فترات حياة المصفوفات المتبقية
U=U-W -- SRAM هو مجموعة كل المصفوفات التي حجمها أصغر من حجم U
While (U≠ ∅)
        For each array u∈ U
```

Find largest clique c(u) in G such that u ∈ c(u) and size(v) ≤ size(u) ∀v ∈ c(u)

Compute access density $\mathbf{AD(u)} = \frac{\sum_{v \in c(u)} \mathbf{TCF(v)}}{\mathbf{size(u)}}$

end for

Assign clique c(i) to SRAM, where AD(i)= max{AD(u)|u∈U}

إسناد الزمرة ذات كثافة الدخول الأعلى إلى SRAM –

–

AvSpace = AvSpace – size(c) -- size(c) هو حجم أكبر مصفوفة في c

X={v∈U|size(v)>AvSpace} -- هو مجموعة U أكبر من AvSpace المصفوفات من X

W=W∪X --SRAM المصفوفات في X التي يتم وضعها في

U=U–{v|(v∈c)}–X --U إزالة SRAM والمصفوفات في X من المجموعة الكلية المصفوفات التي تم إسنادها إلى

end while

Assign array in W to DRAM

end Algorithm

الدخل في هذه الخوارزمية هو حجم SRAM وبرنامج التطبيق P، مع وضع علامة على المتغيرات المخصصة للمسجل. أما الخرج فهو تعيين كل متغير إلى ذاكرة Scratch–Pad أو DRAM.

تقوم الخوارزمية أولاً بتعيين الثوابت والمتغيرات العددية إلى SRAM، والمصفوفات التي تكون أكبر من ذاكرة Scratch–Pad، إلى DRAM. بالنسبة للمصفوفات المتبقية، نقوم أولاً بإنشاء المخطط التوافقي G، حيث تمثل العقد المصفوفات في البرنامج P، وتوجد حافة بين عقدتين إذا كانت المصفوفات المتقابلة لها فترات حياة منفصلة.

المشكلة المماثلة لتوضع المتغيرات العددية في ملف المسجل يتم حلها باستخدام تقسيم الزمرة [18] للمخطط التوافقي، بحيث أن الزمرة هي مخطط فرعي متصل بشكل كامل من المخطط التوافقي وتمثل مسجلًا واحدًا – جميع العقد في الزمرة لها فترة حياة منفصلة ويمكن تعيينها في نفس المسجل. مشكلة تقسيم الزمرة، التي تم تعريفها أنها NP–complete [17]، هي لتقسيم المخطط G إلى عدد أدنى من الزمر المنفصلة – وهذا يقلل من عدد المسجلات المطلوبة لتخزين المتغيرات العددية. لكن ومع ذلك، لا يمكننا تطبيق خوارزمية تقسيم الزمرة بطريقة مباشرة عندما نقوم بتجميع المصفوفات في كتل لأن الزمر التي نهتم بها ليست بالضرورة منفصلة. على سبيل المثال، النظر في زمرة تتكون من ثلاثة مصفوفات A، B وC، حيث حجم A أكبر من حجم B وحجم C. على افتراض أن مساحة المتوفرة حالياً من ذاكرة SRAM أثناء تكرار واحد من التعيين هو AvSpace، الزمرة {A,B,C} لن تتناسب مع SRAM إذا كان حجم A أكبر من AvSpace. ومع ذلك ، فإن المجموعة الفرعية {B,C} ستناسب SRAM إذا كان حجم B وحجم C أصغر من AvSpace. وبالتالي ، نحن بحاجة إلى النظر في كل من الزمر {A,B,C} و {B,C} خلال مرحلة تعيين SRAM.

للتعامل مع إمكانية حدوث تداخل المجموعات، سنقوم بإنشاء مجموعة واحدة (زمرة في المخطط G) c(u) لكل مصفوفة u، الذي يتكون من u وجميع العقد v في المخطط G، مثل ذلك حجم v أصغر أو يساوي حجم u، المخطط الجزئي يتكون من العقدc(u) المتصلة بشكل كامل. وهكذا، لكل مصفوفة u، ونحن نحاول أن نحدد الزمرة مع أقصى كثافة للوصول AD(u) (أي أكبر ∑TCF ، من أجل الحد الأقصى لمساحة الذاكرة المطلوبة= حجم(u)).

ويُنظَر بسهولة إلى هذه المشكلة على أن يكون NP باستخدام الاقتراح حيث TCF(u)=1 و size(u)=1 لكل عقدة u، والاستدلال على الانخفاض من مشكلة الزمرة القصوى [17]. نحن نستخدم المساعدة على الكشف الجشع (greedy heuristic) [19]، الذي يبدأ مع زمرة عقدة واحدة تتكون من عقدة u ويضيف بشكل متكرر الجار v مع الحد الأقصى TCF(v) ،والحجم v أصغر أو يساوي حجم u وتمتلك v حافة مع جميع العقد في زمرة التي شيدت حتى الآن.

بعد توليد الزمر لكل مصفوفة u، نقوم بتعيين صاحب أعلى كثافة للوصول إلى SRAM، وبعد ذلك نقوم بحذف جميع العقد في الزمرة وربط حواف من الرسم البياني G. في حالة وجود أكثر من مجموعة واحدة مع نفس كثافة الوصول، نختار المصفوفة ذات قيمة TCF الأكبر. بعد كل مهمة نقوم بإسنادها إلى SRAM، نقوم بتعيين كافة المصفوفات غير المعينة التي هي أكبر من مساحة SRAM المتاحة إلى DRAM. ثم نقوم بإعادة العملية حتى يتم النظر في جميع العقد في G.

لاحظ أن الخطوة بين الزمر يتم إعادة حسابها في كل تكرار لأن الطبيعة المتداخلة للكتل قد يؤدي إلى تغيير نظام المجموعة الأمثل بعد إزالة زمرة.

تحليل خوارزمية الإسناد إلى الذاكرة من أجل التعقيد الحسابي، نلاحظ أن تحديد أكبر زمرة هو الخطوة المهيمنة حسابياً، مع المساعدة على الكشف الجشع تتطلب $O(n^2)$ من الزمن، حيث n هو عدد المصفوفات في برنامج P. تحديد الزمر لجميع المصفوفات يتطلب$O(n^3)$ من الزمن في كل تكرار. إذا الخوارزمية يمكن أن تكرَّر كحد أقصى O(n) مرة، والتعقيد العام هو $O(n^4)$.

عند تطبيق خوارزمية تعيين الذاكرة إلى التطبيقات العملية، لاحظنا أن عدد المصفوفات ليست بالضرورة صغيرة، ولكن عدد من المصفوفات في برنامج مع أزمنة حياة غير المتقاطعة عادة صغيرة. وبالتالي، تميل الرسوم البيانية المتوافقة لمصفوفات برنامج ما إلى أن تكون متفرقة. هذا على النقيض من الرسم البياني المقابل للتدرجات، والتي عادة ما تكون كثيفة. وهذا يشير إلى أن خوارزمية بحث شامل لتحديد الزمر التي قد تكون مقبولة.

في تنفيذنا، نستخدم بحث شامل إذا كان عدد الحواف في الرسم البياني المتوافق أصغر أو يساوي 2n، حيث n هو عدد العقد، وإلا فإننا سنستخدم الكشف الجشع.

7- التجارب والنتائج:

لقد أجرينا تجارب محاكاة على العديد من الأمثلة المعيارية التي تحدث بشكل متكرر كنواة كود برمجي في التطبيقات المضمنة، لتقييم فعالية خوارزمية تقسيم الذاكرة بين الذاكرتينSPM/ DRAM. سنستخدم ذاكرة SPM حجمها 1Kbyte وذاكرة مخبأة لها ذات الحجم وتستخدم التوضع المباشر (Direct_mapped) و طريقة الكتابة (write_back). من أجل البرهان على أهمية تقنيتنا، قمنا بمقارنة الأداء وقياس العدد الإجمالي لدورات المعالج المطلوبة للوصول إلى البيانات أثناء تنفيذ أمثلة على الخوارزميات والبُنى التالية:

(A) الذاكرة المخبأة للبيانات بحجم 2K: في هذه الحالة لا يوجد ذاكرة SPM في البنية؛

(B) ذاكرة Scratch–Pad بحجم 2K :في هذه الحالة لا توجد ذاكرة مخبأة للبيانات في البنية ، ونحن نستخدم خوارزمية بسيطة بحيث تحدد جميع الأعداد والعديد من المصفوفات المناسبة في ذاكرة SPM والباقي في الذاكرة خارج الرقاقة.

(C) التقسيم العشوائي: في هذه الحالة ، استخدمنا ذاكرة SPM وذاكرة مخبأة كل منهما 1K وتقنية تقسيم بيانات عشوائية ؛

(D) تقنيتنا: استخدمنا هنا ذاكرة مخبأة 1K وذاكرة SPM كذلك 1K وخوارزمية إسناد البيانات (Memory Assign)لتقسيم البيانات .

تم اختيار الحجم 2K في (A) و (B) ، لأن المنطقة التي تشغلها ذاكرة SPM / الذاكرة المخبأة ستكون تقريبًا نفس المساحة التي تشغلها 1K SPM+1K Cache ، متجاهلة دائرة التحكم.

نستخدم ذاكرة مخبأة تستخدم التوضع المباشرة بحجم خط 4 كلمات وأوقات الوصول التالية:

- زمن الوصول إلى كلمة واحدة من SPM هو دورة واحدة؛
- زمن الوصول إلى كلمة واحدة من الذاكرة خارج الرقاقة (في حالة عدم وجود ذاكرة مخبأة) هو 10 دورات ؛
- زمن الوصول إلى كلمة من ذاكرة المخبأة للبيانات في حالة الإصابة hit هو دورة واحدة ؛
- زمن الوصول إلى كتلة من الذاكرة من DRAM خارج الرقاقة في حال وجود ذاكرة مخبأة هو 10 دورات للتهيئة +1* حجم الخط في الذاكرة المخبأة أي 10+1*4=14 دورة.

هذا هو الزمن المطلوب للوصول إلى الكلمة الأولى + الزمن للوصول إلى الكلمات المتبقية (المتجاورة). هذا هو النموذج شائع لحركة الذاكرة المخبأة/ الذاكرة خارج الرقاقة.

7–1 الأمثلة المعيارية

يعرض الجدول الأول قائمة بالأمثلة المعيارية التي أجرينا عليها تجاربنا وخصائصها. يعرض العمودان 1 و 2 الاسم ووصفاً موجزاً للمعايير. يعطي العمودان 3 و 4 عدد المتغيرات العددية والمصفوفة ، على التوالي ، في المواصفات السلوكية. يعطي العمود 5 الحجم الإجمالي للبيانات لكل معيار.

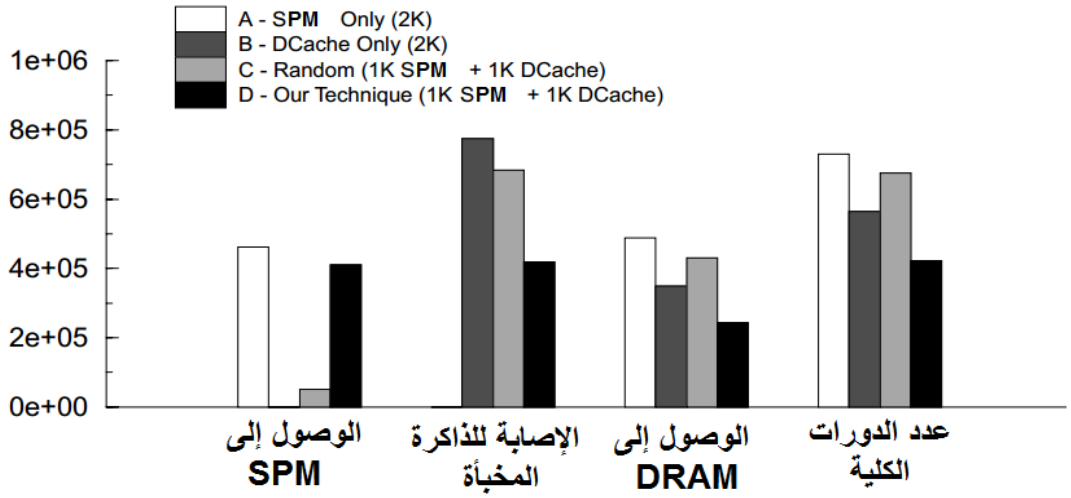
الجدول (1): خصائص الأمثلة المعيارية

Benchmark	Description	No. of Scalars	No. of Arrays	Data Size (Bytes)
Beamformer	Radar Application	7	7	19676
Dequant	Dequantization Routine (MPEG)	7	5	2332
FFT	Fast Fourier Transform	20	4	4176
IDCT	Inverse Discrete Cosine Transform	20	3	1616
MatrixMult	Matrix Multiplication	5	3	3092
SOR	Successive Over-Relaxation	4	7	7184
DHRC	Differential Heat Release Computation	28	4	3856

7–2 تطبيق الرادار Beamformer

يوضح الشكل (7) تفاصيل الوصول إلى الذاكرة للمثال المعياري Beamformer. نلاحظ أن التشكيل (A) الموضح سابقاً لديه أكبر عدد من عمليات الوصول إلى ذاكرة SPM (SPM Accesses) لأن ذاكرة

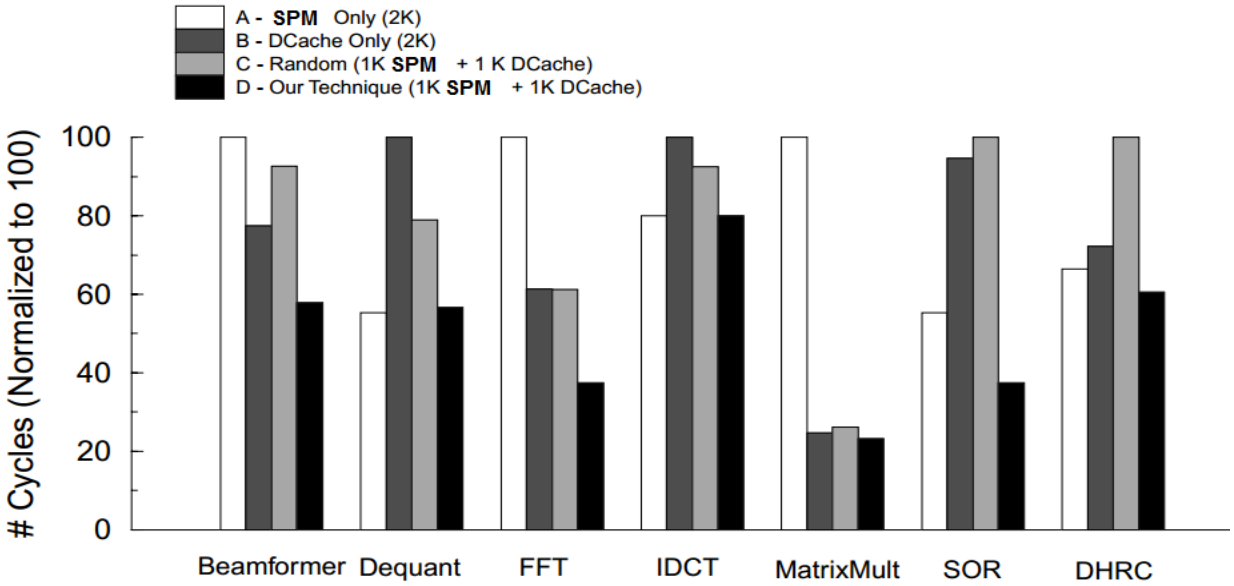
SPM الكبيرة (2 كيلو بايت) تسمح بتعيين المزيد من المتغيرات في ذاكرة SPM. أما في التشكيل (B) لا يوجد وصول إلى SPM، نظراً لعدم وجود SPM في هذا التشكيل. أيضاً ، ينتج عن تقنيتنا (D) وصول إلى SPM أكثر بكثير من تقنية التقسيم العشوائي لأن التقنية العشوائية تتجاهل السلوك عندما تقوم بتعيين مساحة SPM . وبالمثل ، فإن نتائج الإصابة في الذاكرة المخبأة (Cache Hits) هي الأعلى في B، والصفر في A. ينتج عن تقنيتنا عدد مرات إصابة الذاكرة المخبأة أقل منها في التشكيلة C لأنه في C العديد من عناصر الذاكرة التي يتم الوصول إليها من خلال الذاكرة المخبأة، ويتم تعيينها في ذاكرة SPM في تقنيتنا. التشكيل A لديه عدد مرتفع من الوصول إلى ذاكرة (DRAM) (DRAM Accesses) لأن غياب الذاكرة المخبأة يسبب أن كل وصول إلى البيانات يحتاج إلى الوصول إلى الذاكرة الخارجية DRAM. نتيجة لذلك، نلاحظ أن العدد الإجمالي لدورات المعالج المطلوبة للوصول إلى جميع البيانات (Total Cycles) هو الأعلى بالنسبة للتشكيل A التشكيل D ينتج أسرع زمن وصول، بسبب رسم الخرائط للعناصر الأكثر تعرضًا للوصول إلى SPM.



الشكل (7) تفاصيل الوصول إلى الذاكرة للمثال المعياري Beamformer

7-3 أداء تشكيلات الذاكرة المستندة إلى SPM

يقدم الشكل (8) مقارنة لأداء التشكيلات الأربعة A و B و C و D الموضحة سابقاً، على نواة الكود المستخرجة من سبعة تطبيقات مضمنة للمعايير السابقة. بحيث عدد الدورات لكل تطبيق 100. في مثال Dequant ، يتفوق A قليلاً على الأداء في D لأن جميع البيانات المستخدمة في التطبيق تتناسب مع 2K SPM المستخدمة في A، بحيث الوصول إلى الذاكرة خارج الرقاقة ليس ضرورياً على الإطلاق، مما يؤدي إلى أداء أسرع. ومع ذلك، فإن حجم البيانات التي تكون أكبر من 1K SPM المستخدمة في D، حيث يتسبب الفقد في الذاكرة المخبأة الإلزامية إلى حدوث تدهور طفيف في الأداء. تظهر نتائج FFT و MatrixMult ، كلا التطبيقين يستخدمان في الحوسبة إلى أن A عبارة عن تشكيل ادنى لتطبيقات الحوسبة الموجهة القابلة للتعديل لاستثمار المنطقة المرجعية. تؤدي تعارضات الذاكرة المخبأة إلى انخفاض أداء B و C في SOR و DHRC ، مما يتسبب في أداء أسوأ من A (حيث لا توجد ذاكرة مخبأة) ، و D (حيث يتم تقليل التعارضات عن طريق الخوارزمية تعيين نوع الذاكرة).



الشكل (8) مقارنة لأداء التشكيلات الأربعة A و B و C و D على نواة الكود المستخرجة من سبعة تطبيقات مضمنة للمعايير

أسفرت تقنيتنا عن تحسن متوسط بنسبة 31.4٪ من أجل A و 30.0٪ من أجل B و 33.1٪ من أجلC.

وباختصار ، فإن تجاربنا على شيفرات من تطبيقات النظم المضمنة النموذجية تُظهر فائدة ذاكرة Scratch–Padعلى الرقاقة، بالإضافة إلى الذاكرة المخبأة للبيانات بالإضافة إلى فعالية تقنية تقسيم البيانات.

8- الاستنتاجات

تستخدم تطبيقات النظم المضمنة الحديثة نوى المعالج جنباً إلى جنب مع الذاكرة وأجهزة المعالج الأخرى على الشريحة نفسها. فمن المهم الاستفادة المثلى من المنطقة على الرقاقة، من أجل استخدام الذاكرة على الشريحة بشكل فعال ، نحتاج إلى الاستفادة من مزايا كل من الذاكرة المخبأة للبيانات (عنونة بسيطة) وذاكرة Scratch–Padعلى الشريحة (ضمان وقت وصول منخفض) من خلال تضمين كلا النوعين من الذاكرة في نفس الشريحة، مع تقسيم مساحة ذاكرة البيانات بشكل منفصل بين الاثنتين.

قدمنا استراتيجية لتقسيم المتغيرات (الأعداد والمصفوفات) في التعليمات البرمجية المضمنة في ذاكرة Scratch–Padوالذاكرة المخبأة للبيانات التي تحاول تقليل تعارضات الذاكرة المخبأة للبيانات. تُظهر تجاربنا على نواة التعليمات البرمجية من التطبيقات النموذجية تحسينات كبيرة في الأداء (انخفاض بنسبة 33 إلى 33٪ في وقت الوصول إلى الذاكرة) مقارنة بالبنى المماثلة واستراتيجيات التقسيم العشوائي.

المراجع

[1] Benini, L. ; Macii, A; Macii, E.; Poncino, M., *"Increasing Energy Efficiency of Embedded Systems by Application-Specific Memory Hierarchy Generation"*, IEEE Design and Test of Computers, Volume 17 Issue 2, Apr-Jun 2000, pp. 74-85, 2000.

[2] LANGGUTH,D.2008 Scratchpad memory vs Caches Performance and Predictability comparison.

[3] Preeti Ranjan Panda, Nikil D. Dutt, Alexandru Nicolau ,*"Memory Issues in Embedded Systems-on-Chip:Optimizations and Exploration",SPRINGER SCIENCE+BUSINESS MEDIA*,LLC 2012.

[4] LSI Logic Corporation, "CW33000 MIPS Embedded Processor User's Manual," 1992.

[5] Rajeshwari Banakar, Stefan Steinke, BS Bo-Sik Lee, M. Balakrishnan, *and Peter Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In Proceedings of the . . .* , page 73, New York, New York, USA, May 2002. ACM Press.

[6] LEE,E.A ; SESHIA,S.A. *INTRODUCTION TO EMBEDDED SYSTEMS A CYBER-PHYSICAL SYSTEMS APPROACH. 2nd .ed*, MIT Press, Cambridge ,Massachusetts, USA, 2017, 585.

[7] Matthew Kuo, *Partha Roop, Sidharta Andalam, and Nitish Patel. Precision Timed Embedded Systems Using TickPAD Memory*. In 2013 13th Int. Conf. Appl. Concurr. to Syst. Des. IEEE, 2013.

[8] Sidharta Andalam, Partha Roop, *Alain Girault, and Claus Traulsen. PRET-C: A new language for programming precision timed architectures. Technical report*, 2009.

[9] Nan Guan, Wang Yi, Qingxu Deng, Zonghua Gu, and Ge Yu. Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling. J. Syst. Archit., 2011.

[10] David Tam, Reza Azimi, Livio Soares, *and Michael Stumm. Managing shared L2 caches on multicore systems in software*. In Work. Interact. between Oper. Syst. Comput. Archit., 2007.

[11] WASLY,S. "*Scratchpad Memory Management For Multicore Real-Time Embedded Systems"*. A thesis presented to the University of Waterloo in fulfillment of the thesis requirement for the degree of Doctor of Philosophy in Electrical and Computer Engineering Waterloo, Ontario, Canada, 2018.

[12] WASLY,S. 2012, *A Dynamic Scratchpad Memory Unit for Predictable Real-Time Embedded Systems*. A thesis presented to the University of Waterloo in fulfillment of the thesis requirement for the degree of Master of Applied Science in Electrical and Computer Engineering, Waterloo, Ontario, Canada.

[13] ROOB,J. " *An Introduction to the Research on Scratchpad Memory: Definition, Hardware*, Known Implementations and WCET Optimisation". Embedded Systems Group, University of Kaiserslautern Germany, 2013.

[14] ANURADHA,B; VIVEKANANDAN." *Usage of scratchpad memory in embedded systems*". IEEE, INDIA, JULY 2012.

[15] STALLINGS,*W. Computer Organization And Architecture Designing For Performance. 9th .ed*, PEARSON, United States of America,2013,787.

[16] P. Ranjan Panda ; H. Nakamura ; N.D. Dutt ; *A. Nicolau A data alignment technique for improving cache performance IEEE Austin*, TX, USA, USA 06 August 2002.

[17] Brian C. Dean, *Michel X. Goemans, Jan Vondrák Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity* , ieee ,3 Nov 2008.

[18] Solving group technology problems via clique partitioning Haibo Wang, *Bahram Alidaee, Fred Glover & Gary Kochenberger International Journal of Flexible Manufacturing Systems volume 18*, pages77–97(2006).

[19] Greedy heuristic method for location problems L. A. Kazakovtsev*, *A. N. Antamoshkin Reshetnev Siberian State Aerospace University* , Russian Federation Vestnik SibGAU Vol. 16, No. 2, P. 317-325 ,July 2015.