معالجة مشكلة اتساق الكاش في الأنظمة المتعددة المعالجات

د. م. جمال الياسين[1]

الملخص

تحتاج الأنظمة الحاسوبية (الأنظمة المتعددة المعالجات) إلى تجاوز مشكلة اتساق الكاش. وقد حلّت هذه المشكلة في الأنظمة المتعددة المعالجات الحديثة بتزويدها ببروتوكولات اتساق الكاش. تؤثر بروتوكولات اتساق الكاش في أداء الأنظمة المتعددة المعالجات ذات الذاكرة المشتركة. في هذه الدراسة سنناقش عدد من بروتوكولات اتساق الكاش المختلفة بما يتضمن ميزات كل منها ومساوئه، والطرائق التي ينظّم بها عمل هذه البروتوكولات، وأمثلة عن بعض الأنظمة المزودة بهذه البروتوكولات.

كلمات مفتاحية: اتساق الكاش – بروتوكولات الاستطلاع – بروتوكولات الدليل – بروتوكول الـbit-vector –بروتوكول الـ coarse-vector بروتوكول تخصيص المؤشر الديناميكي – بروتوكول الواجهة المتدرجة المتماسكة.

[1] أستاذ مساعد – قسم هندسة الحواسيب والأتمتة – كلية الهندسة الميكانيكية والكهربائية – جامعة دمشق.

Addressing the Problem of Cache Coherence in Multiprocessor Systems

Dr. Eng. Jamal Al Yasin[(1)]

Abstract

Computer systems (multi processors) need to avoid the cache coherence problem which has been solved by today's multiprocessors through implementing a cache coherence protocol. The cache coherence protocol affects the performance of a distributed memory-shared multiprocessor system. This paper discusses several different varieties of cache coherence protocols including advantages and disadvantages, the way they are organized, and some examples of systems that implement those protocols.

Keywords: Cache Coherence - Snoopy Protocol - Directory Protocol – Bit-vector Protocol – Coarse-vector Protocol - Dynamic Pointer Allocation Protocol - Scalable Coherent Interface Protocol.

[(1)] Department Automation and Computer Engineering, Faculty of Mechanical & Electrical Engineering-Damascus University

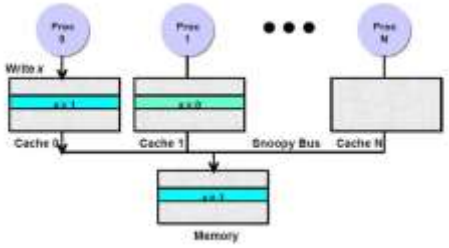
المقدمة:

أدى وجود ذواكر الكاش في الأجيال الحالية من الأنظمة متعددة المعالجات الموزعة ذات الذواكر المشتركة **(distributed shared-memory multiprocessors)** إلى تحسين الأداء بتخفيض الزمن اللازم للوصول إلى الذاكرة فضلاً عن تقليل متطلبات عرض الحزمة اللازمة للاتصال بين كل من الذاكرة المحلية، والمعالج، والاتصال العام بين المعالجات. لكن لسوء الحظ فإنّ وجود ذاكرات كاش محلية أدى إلى ظهور مشكلة اتساق الكاش. في الأنظمة الموزعة الأولى ذات الذاكرة المشتركة كانت تحلّ هذه المشكلة حلاً برمجياً من قبل المبرمجين. أمَّا اليوم فأصبح الحل لهذه المشكلة عتادياً (hardware)، ذلك بابتكار بروتوكولات اتساق الكاش. في هذا البحث طرحت مشكلة اتساق الكاش، ونوضّح كيف تقوم بروتوكولات اتساق الكاش بحل هذه المشكلة، كما قدّم شرح تفصيلي لعدد من بروتوكولات اتساق الكاش المختلفة، ونوضّح ميزات كل منها ومساوئه.

1. مشكلة اتساق الكاش:

في نظام متعدد المعالجات ذي ذاكرة مشتركة وذاكرات كاش منفصلة بحيث يكون لكل معالج من هذه المعالجات ذاكرة خاصة به، يمكن أن يوجد أكثر من نسخة لمعامل إحدى التعليمات: نسخة موجودة في الذاكرة الرئيسة المشتركة، ونسخة في كل ذاكرة كاش من كاشات المعالجات. في حال تغيرت إحدى نسخ هذا المعامل يجب على النسخ الأخرى جميعها الموجودة في ذواكر الكاش للمعالجات الأخرى أن تتغير أيضاً. من ثمّ النظام ذو اتساق الكاش هو النظام الذي يؤمن نشر تغير قيمة هذا المعامل المشترك على ذواكر النظام جميعها في الوقت المناسب.

يصف الشكل (1) مثال لمشكلة اتساق الكاش. تحوي الذاكرة في البداية في الموقع x القيمة 0 ولدينا معالجان (معالج 0 ومعالج 1) كلاهما يقرأ الموقع x، ويضعه في ذاكرة الكاش الخاصة به. إذا قام المعالج 0 بكتابة القيمة 1 في الموقع x في ذاكرة الكاش الخاصة به تكون قيمة الموقع x في ذاكرة الكاش الخاصة بالمعالج 1 لاتزال 0. في القراءة التالية لقيمة الموقع x من قبل المعالج 1 سوف يقوم بإعطاء القيمة السابقة المخزنة، وهي 0. ولكن ذلك لا يطابق ما أراده المبرمج عندما قام بكتابة برنامجه حيث القيمة المتوقعة هي آخر قيمة تم كتابتها للمتحول x (وهي في حالتنا تساوي 1). وهذا بالضبط ما تفعله بوتوكولات اتساق الكاش: تؤمن أنّه عند طلب قيمة موقع ما سوف تعطي القيمة الأحدث لهذا الموقع.[1]



الشكل (1) مشكلة اتساق الكاش

تقوم بروتوكولات اتساق الكاش بتحقيق هذا الهدف من خلال اتخاذ إجراءات عندما يتم إعادة الكتابة في موقع معين في ذاكرة كاش أحد المعالجات. يمكن للبروتوكولات أن تتخذ نوعين من الإجراءات عندما يتم إعادة كتابة الموقع x:

1- إمَّا أن تبطل صلاحية كل نسخ x في ذواكر الكاش جميعها.

2- أو تقوم بتعديل قيم هذه النسخ وإعطائها القيمة الجديدة.

بالعودة إلى المثال السابق في بروتوكولات إبطال الصلاحية عندما يكتب المعالج 0 القيمة 1 في xتصبح قيمة x غير صالحة في كاش المعالج 1 ومن ثمّ في المرة

التالية التي يطلب فيها المعالج 1 قراءة قيمة x سوف تؤدي إلى حدوث إخفاق (cash miss)، وستطلب القيمة الجديدة من الذاكرة الرئيسة المشتركة. في الأنظمة التي تتبع نمط الـwrite-through يمكن للذاكرة الرئيسة أن تزود المعالج 1 بقيمة x لأنّه تم كتابة القيمة الجديدة لـx مباشرة بعد تعديلها من قبل المعالج 0. أمّا في أنظمة الـwrite-back فيجب على بروتوكول اتساق الكاش أن يتحقق من أن المعالج 1 طلب القيمة الأحدث من المعالج 0. ومن ثمّ يقوم المعالج 0 بتزويد قيمة x لذاكرة كاش المعالج 1 بعد حدوث إخفاق ذاكرة (cash miss). أمّا في حال البروتوكولات التي تعتمد على تحديث نسخ المعطيات المعدلة جميعها في ذواكر الكاش جميعها فعندما يقوم المعالج 0 بكتابة x=1 سوف يرسل النسخة الجديدة من المعطيات مباشرة إلى المعالج 1 و يحدث قيمة x فيه إلى القيمة الجديدة.[2][1]

تستخدم معظم أنظمة المعالجات المتعددة ذات اتساق الكاش الحديثة بروتوكولات إلغاء الصلاحية بدلاً عن بروتوكولات تحديث القيم.

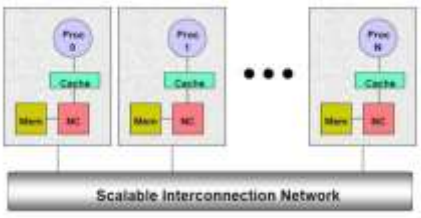
اذ كلما ازداد عدد المعالجات (طول خط ذواكر الكاش) أصبح استخدام خوارزميات إلغاء الصلاحية شائعاً أكثر بسبب ازدياد عدد التعديلات المطلوبة في حال استخدام بروتوكولات التعديل المباشر.

في بعض الأحيان يفضل استخدام بروتوكولات التعديل ،ولكن ذلك يؤدي إلى عدد أكبر من مرات الوصول إلى الذاكرة فضلاً عن مشكلات في اختلاف التزامن بين الذواكر. نموذجياً يختار المصممون بروتوكولات إلغاء الصلاحية ويضيفون اليها بعض الخصائص لتستطيع التوافق مع اختلافات التزامن

2. بروتوكولات الاستطلاع وبروتوكولات الدليل:

تقسم بروتوكولات اتساق الكاش إلى صنفين أساسيين: بروتوكولات الاستطلاع ((snoopy protocol، وبروتوكولات الدليل (directory protocol).

1.2. بروتوكولات الاستطلاع: تحتاج بروتوكولات الاستطلاع وسط بث (broadcast medium) ومن ثمّ تطبّق فقط في الأنظمة متعددة المعالجات ذات الناقل المشترك التي يكون فيها عدد المعالجات قليلاً. في نظام البث هذا تقوم كل ذاكرة كاش باستطلاع حالة الناقل، ومراقبة تغيرات المعلومات (data) التي يستشعرها هذا الناقل. في أي وقت تستطلع فيه ذاكرة الكاش من خلال الناقل المشترك حدوث عملية كتابة، تقوم بإلغاء تفعيل قيمة معامل الذاكرة الذي عدّل إذا كان موجوداً فيها. في أي وقت يصل طلب قراءة إلى الكاش عن طريق الناقل، تقوم الكاش بالتأكد من أنّ المتحول المطلوب قراءته قيمته تساوي قيمة آخر تحديث له، وفي حال كان ذلك محققاً يقوم بالإجابة عن طلب الناقل. يعدّ هذا البروتوكول سهل البناء ولكن لسوء الحظ في حال ازدياد عدد المعالجات يصبح الناقل موضع اختناق (عنق الزجاجة). لمعالجة هذه المشكلات، اعتمد المصممون بنية الذاكرة المشتركة الموزعة (DSM). كل عقدة في نظام DSM متعدد المعالجات تحوي معالجاً وذواكر الكاش الخاصة به، اذ يقوم جزء من الذاكرة الموزعة الرئيسة للنظام ومتحكم خاص بالعقدة بتنظيم الاتصال بين العقد (الشكل 2).

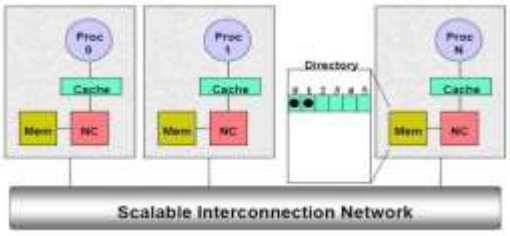


الشكل (2) مثال عن نظام ذي ذاكرة مشتركة موزعة.

بدلاً من أن يكون الاتصال بين العقد عن طريق ناقل وحيد يصبح الاتصال عن طريق شبكة اتصال بيني. يسمح نظام ال– DSM بربط عدد كبير من العقد يصل إلى آلاف العقد، ولكن عدم وجود وسط بث (broadcast medium) يشكل مشكلة لبروتوكولات اتساق الكاش. ومن ثمّ بروتوكولات الاستطلاع Snoopy لم تعد مناسبة، وبدلا من ذلك على المصممين أن يستخدموا بروتوكول الدليل.[3]

2.2. بروتوكولات الدليل:

الدليل هو ببساطة بنية معطيات مساعدة تقوم بالتقاط حالة كل ذاكرة كاش من خط كاشات النظام. من أجل كل خط كاش (cache line) (يقصد بخط الكاش: عنصر المعطيات الموجود نسخ منه في عدة ذواكر كاش)، على الدليل أن يعرف أي ذاكرة كاش لديها نسخة قراءة فقط، أو أي ذاكرة من الخط لديها آخر تحديث. تعمل الأجهزة التي تطبق فيها بروتوكولات الدليل على استشارة الدليل عند حدوث أي حالة فقد (miss)، وتقوم بالفعل المناسب اعتماداً على نمط الطلب وحالة الدليل الحالية. ويبيّن الشكل (3) آلة DMS معتمدة على بروتوكولات الدليل.[2]



الشكل (3) نظام ذو ذاكرة مشتركة موزعة مطبق عليه بروتوكول الدليل لاتساق الكاش

بشكل مشابه للتوزيع الفيزيائي للذاكرة الرئيسة على العقد و من أجل تحسين عرض حزمة الذاكرة الإجمالي، وزّع الدليل لمنع حدوث اختناق عنق الزجاجة (وهو المصطلح الذي يطلق على الحالة التي تؤدي إلى هبوط تدني الأداء أو تدنيه بسبب ضعف أحد أجزاء الـ Hardware). إذاً قسّمت الذاكرة الرئيسة في كل عقدة إلى كتل حجمها يساوي حجم خط الكاش(line–cache)، ومن ثمّ يستطيع الدليل أن يضيف بتات حالة الى كل بلوك من الذاكرة الأساسية. عندما يحتاج المعالج لقراءة خط الذاكرة L، يجب عليه أن يرسل طلباً إلى العقدة التي تحوي دليل هذا الخط L. تدعى هذه العقدة بالعقدة الرئيسة بالنسبة الى L. تستقبل العقدة الرئيسة الطلب، وتشاور الدليل، وتتخذ الإجراء المطلوب. عند حدوث إخفاق الكاش، على سبيل المثال، إذا أظهر الدليل أنّ الخط حالياً غير موجود في ذواكر الكاش، أو موجود ولكن بنمط القراءة فقط (يمكن عدّ الخط نظيفاً clean)، حيث تقوم العقدة الرئيسة بجعل العقدة الطالبة كجزء من الدليل وتجيب العقدة الطالبة بنسخة من الخط L من الذاكرة الرئيسة الخاصة بها. إذا أظهر الدليل أن هناك عقدة ثالثة تحوي هذه المعطيات (data) ولكن تم التعديل عليها في ذاكرة الكاش الخاصة بها (الخط متسخ dirty)، تحول العقدة الرئيسة الطلب إلى هذه العقدة الثالثة، وهي بدورها تكون المسؤولة عن إحضار المعطيات من ذاكرة الكاش الخاصة بها، وإرسالها الى العقدة الطالبة، وترسل هذه العقدة (العقدة الثالثة) رسالة للعقدة الرئيسة لتخبرها بنجاح عملية النقل. إنّ المثال المبسط المشروح أعلاه يبين للقارئ مدى تعقيد تنفيذ بروتوكول يحافظ على اتساق الكاش بشكل كامل في آلة ذات نظام الذاكرة المشتركة الموزعة والدليل الموزع.

يوجد في بروتوكولات اتساق الكاش المعتمدة على الدليل مكونان أساسيان:

* تنظيم الدليل (the directory organization)
* أنماط الرسائل وتأثيراتها

(message types and message actions)

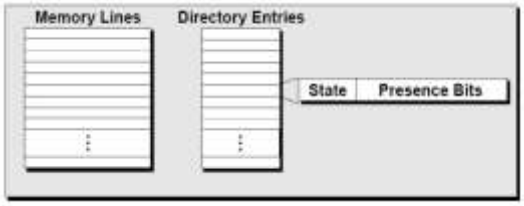
ينظّم الدليل من خلال بنية المعطيات المستخدمة لتخزين معلومات الدليل، وهي تؤثر تأثيراً مباشراً في البتات المستخدمة لتخزين المعلومات عن كل خط كاش. الذاكرة المطلوبة لحفظ معلومات الدليل هي مصدر قلق لأنّها ذاكرة إضافية(extra memory) غير مطلوب توفرها من أجل الأنظمة التي لا تستخدم بروتوكولات الدليل. نسبة كمية الذاكرة اللازمة للدليل على كمية الذاكرة المستخدمة في النظام تدعى directory memory overhead. يسعى المصمم إلى جعل هذه النسبة أصغر ما يمكن. تحوي ذاكرة تنظيم الدليل حالة بروتوكول الدليل، وكن على البروتوكول أن يرسل رسائل من العقد واليها لإيصال تغيرات حالة البروتوكول، معطيات الطلبات والمعطيات المعادة. كل رسالة بروتوكول مرسلة عبر الشبكة لها نمط معين أو صيغة معينة opcode مرتبطة بها، كما تقوم كل عقدة بعمل معين اعتماداً على نمط الرسالة التي استقبلتها وآلة النظام الحالية. وكذلك تتضمن مجموعة الأعمال الناتجة عن الرسائل قراءة حالة الدليل وتعديلها حسب الحاجة، معالجة حالات السابقة كلّها، معالجة الحالات العابرة كلّها، تشكيل أي رسالة إجابة للطلب response message، وغيرها من الأعمال.

أنشئت أهم ثلاثة بروتوكولات مختلفة من بروتوكولات الدليل للعمل على أجهزة DMS الصناعية، كما طرحت بروتوكولات أخرى في مجموعة البحوث. كل بروتوكول يختلف عن غيره من حيث تنظيم الدليل (ومن ثمّ ذاكرة الدليل المضافة)، وعدد الرسائل المتبادلة بين العقد وأنماطها.[4]

1.2.2. بروتوكولا الـBit-vector و-Coarse vector:

صمم بروتوكول الـbit-vector ليكون سريعاً وفعّالاً في الآلات قليلة و متوسطة عدد المعالجات، وهو الأبسط من بين بروتوكولات اتساق الكاش جميعها.

يبيّن الشكل (4) مثالاً عن تنظيم بروتوكول الدليل bit-vector.



الشكل (4) بنية معطيات بروتوكولي الـbit vector و الـcoarse vector.

لكل خط كاش في الذاكرة الرئيسة يترك هذا البروتوكول directory entry التي تحتفظ بمعلومات الحالة الضرورية كلّها لذلك الخط. معظم مدخلات الدليل مخصصة لسلسلة من بتات الوجود، ومن هنا جاءت تسمية هذا البروتوكول. بت التواجد يأخذ القيمة 1 إذا كانت العقدة الموافقة له تحوي حالياً نسخة من خط الكاش، وماعدا ذلك تكون قيمته 0. أمّا البتات المتبقية من مدخلات الدليل فهي بتات حالة تدلّ إذا ما كان الخط dirty، في حالة انتظار، أو في نظام الدخل / خرج، وغيرها من حالات التنفيذ المحتلمة. في الأنظمة التي تملك عدداً كبيراً من المعالجات، P، زيادة عدد بتات الوجود بصبح محرماً لأنentry~ directory يصبح عرضه كبيراً جداً.[2]

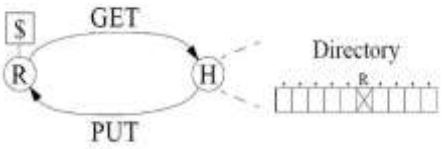
لتوسيع نطاق بروتوكول الـbit-vector ليناسب الآلات ذات عدد المعالجات الكبير يمكن تحويل بروتوكول الـbit-vector إلى بروتوكول الـcoarse-vector. يجري هذا التحويل بشكل دقيق جداً. لنفرض المثال التوضيحي الآتي: اذ بروتوكول الـbit-vector يحوي 48 بت وجود. في

بروتوكول الـ –coarse vector، من أجل نظام عدد معالجاته بين 49 و 96، كل بت فيه يمثل في الـ-bit vector عقدتين، من أجل نظام عدد معالجاته بين 97و192 معالج كل بت من الـbit-vector يمثل أربع عقد و هكذا. خشونة البروتوكول تعرف بعدد العقد التي يمثلها كل بت في تمثيل الـbit-vector. بروتوكول الـ bit vector له خشونة 1. إذا كان بـ48 بت وجود يمكن تمثيل 64 معالجاً ومن ثمّ الخشونة تساوي 2. و128 معالجاً لها خشونة 4. من أجل الـcoarse vector، بت الوجود يكون بحالة وضع (set) إذا كان أي من العقد الممثلة بذلك البت تتشارك في خط الكاش.

تعدّ بروتوكولات الـbit-vector و coarse-vector بشكل مفاهيمي بسيطة، وهي السبب الأساسي لشعبيتها وتطبيقها في العديد من الأجهزة، منها آلة Stanford DASH التي تحوي 64 معالجاً وآلة HaL-S1 التي تحوي 16 معالجاً. مع أنّ هذين الجهازين يستخدمان بروتوكول الbit-vector على مستوى الدليل إلاّ أنّ كليهما يحوي على خشونة ضمنية بمقدار 4، لأن كل بت من مدخلات الدليل تعبّر عن عقدة واحدة التي بلورها تحوي 4 معالجات متعددة متناظرة Symmetric Multiprocessor (SMP). في كل من الآلتين السابقتين يستخدم بروتوكول snoopy للحفاظ على اتساق الكاش داخل العنقود، وبروتوكول bit-vector يستخدم للحفاظ على اتساق الكاش بين العناقيد.[3]

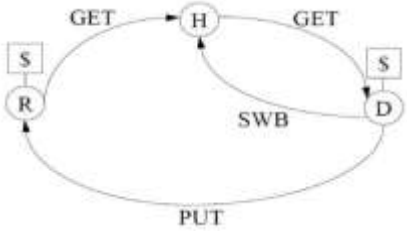
1.1.2.2. إخفاق القراءة في بروتوكولي Bit-vector/Coarse-vector:

في حال حدوت إخفاق قراءة الكاش في أحد المعالجات، يرسل طلب قراءة (GET) إلى العقدة الرئيسة home node لقراءة العنوان الذي حدث عنده الفقد. عند وصول الطلب، تبحث العقدة الرئيسية في مدخلات الدليل عن السطر المطلوب. إذا وجد الدليل أنّ سطر الكاش غير موجود في ذواكر الكاش، أو أنّه موجود، ولكن بنمط القراءة فقط بأي عدد من المعالجات سوف يقوم بروتوكول الـ bit-vector بالأداء نفسه، وهو موضّح بالشكل (5).



الشكل (5) إخفاق القراءة لخط كاش غير محمل في ذواكر الكاش.

تقوم العقدة الرئيسة بإعطاء القيمة 1 لبت الوجود presence الموافق لرقم العقدة الطالبة ويجيب طلب القراءة بالمعطيات المطلوبة الموجودة في الذاكرة الرئيسة من خلال رسالة PUT. إذا كان الـbit dirty في حالة وضع (قيمته تساوي 1) في مدخلات الدليل، من ثمّ يمكن لمعالج واحد فقط أن يكون بت الوجود الخاص به قيمته الـ1 . إذا كان الـbit dirty بحالة وضع تقوم العقدة الرئيسة بإعادة إرسال طلب القراءة إلى مالك خط الكاش كما هو موضّح بالشكل (6).



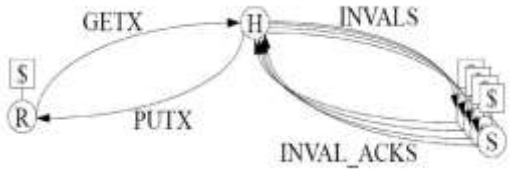
الشكل(6) إخفاق قراءة ذو ثلاث قفزات مع إعادة الارسال.

عندما تستقبل الـdirty node طلب القراءة المعاد إرساله، تسترجع الـdirty data من ذاكرة الكاش الخاصة بها، وتجعل المعلومات الموجودة فيها في حالة مشاركة وتقوم بعملين. العمل الأول، تقوم بإجابة العقدة الطالبة بسطر الكاش المطلوب. الثاني، ترسل رسالة(sharing write back (SWB إلى العقدة الرئيسة. بعد استقبال

هذه الرسالة يصبح لدى العقدة الرئيسة علم بأنّ عملية النقل قد تمت بنجاح من ثمّ تقوم بتصغير الـdirty bit وإعطاء القيمة 1 لبت الوجود الخاص بالعقدة الطالبة R وكتابة المعطيات التي عدّلت في الذاكرة الرئيسة. الآن أصبحت حالة الدليل وكأن الخط المشترك في حالة قراءة فقط بين عقدتين، العقدة الطالبة R، والعقدة المالكة المجيبة D، والذاكرة تملك آخر نسخة من الخط.

2.1.2.2. إخفاق الكتابة في بروتوكولي Bit-vector/Coarse-vector

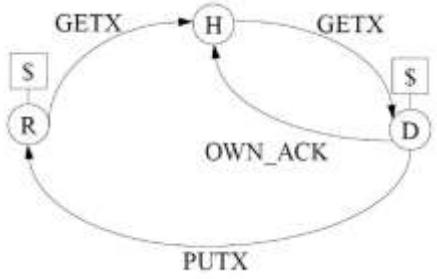
إن عملية طلب الكتابة من قبل المعالج هي أصعب قليلاً من عملية طلب القراءة. في حال إخفاق الكتابة، يرسل طلب الكتابة إلى العقدة الرئيسة التي تشاور الدليل وتعالج واحدة من ثلاث حالات. في الحالة الأولى يمكن للسطر أن يكون غير محتوى في أي ذاكرة كاش من ذواكر النظام. هذه هي الحالة البسيطة. تقوم العقدة الرئيسة بوضع (إعطاء القيمة 1) للـdirty bit، جعل قيمة بت الوجود الخاص بالعقدة الطالبة مساوي للـ1 وإرسال المعلومات (الداتا) المطلوبة الموجودة في الذاكرة الرئيسة. بالتمثيل الصوري عمليات المناقلات تشبه تماماً إخفاق القراءة الممثل بالشكل 5، ولكن باستبدال برسالة الـ GET رسالة GETX (GET exclusive) ورسالة الـPUT برسالة PUTX (PUT exclusive). في الحالة الثانية، الخط يمكن أن يكون مشتركاً بين معالجات عدّة. كما هو موضّح بالشكل (7) . في هذه الحالة العقدة الرئيسة يجب أن ترسل رسالة عدم صلاحية لكل عقدة اعتماداً على مجموعة بتات الوجود، و تصفير بتات اعتماداً على ذلك.



الشكل (7) إخفاق كتابة في خط كاش مشترك.

عندما تكون قد أرسلت رسائل عدم الصلاحية كلّها، تعطي العقدة الرئيسة القيمة 1 للـdirty bit وبت الوجود الموافق للمالك الجديد R اعتماداً على تطبيق بروتوكول معين، أمّا العقدة الرئيسة فتجمع الردود على رسائل عدم الصلاحية invalidation acknowledgments وعندما تستقبلهم كلهم ترسل الداتا إلى العقدة الطالبة، أو العقدة الرئيسة فتعيد إرسال الداتا المطلوبة إلى العقدة الطالبة الرئيسة، وتعطي العقدة الطالبة المسؤولية عن تجميع ردود رسائل عدم الصلاحية.

الحالة الثالثة تماماً مثل حالة الـdirty read case، العقدة الرئيسة تعيد إرسال طلب الكتابة إلى الـdirty third node كما هو موضّح في الشكل (8).



الشكل (8) إخفاق كتابة بثلاث قفزات.

عندما تستقبل العقدة الثالثة dirty node طلب الكتابة، تقوم بجلب البيانات من ذاكرة الكاش الخاصة بها، وتترك خط الكاش غير فعّال. من جديد كما في حالة الـ dirty read العقدة الثالثة البعيدة تقوم بعملين. الأولى ترجع الـdirty cache line إلى العقدة الطالبة، الثانية، ترسل رسالة (ACK–OWN ownership transfer) إلى العقدة الرئيسة. بعد استلام رسالة المالك تبلغ العقدة الرئيسية بأنّ عملية النقل قد تمت بنجاح، تقوم بتصفير بتات الوجود للمالك القديم D وتضع القيمة 1 في بت الوجود في العقدة الطالبة الأساسية R التي أصبحت الآن المالك لخط الكاش.

تكون الانتقالات للبروتوكولات bit-vector protocol and the coarse-vector protocol متماثلة، ما عدا تفضيلاً واحداً لبروتوكول coarse-vector في حالة طلب الكتابة. إنّ بروتوكول coarse-vector هو البروتوكول الوحيد الذي يترك معلومة غير دقيقة – الدليل لا يعرف بالضبط أي معالج يقوم بمشاركة خط معطيات معين. عندما يتم كتابة خط كاش يجب أن يرسل المتحكم رسالة عدم صلاحية لكل معالج يكون البت الموافق له يساوي 1 في شعاع البتات. في بروتوكول الـ coarse vector كل بت يمثل أكثر من معالج واحد ،والمعالج الرئيسي عليه أن يرسل رسائل عدم صلاحية أيضاً، ويستقبل رسائل عدم صلاحية من كل معالج من المجموعة بغض النظر هل المعالج يحوي السطر في ذاكرة الكاش الحاصة به أم لا ؟ وهذا يمكن أن يولد زيادة في الازدحام نتيجة هذه الرسائل بالنسبة الى بروتوكول يحافظ على معلومات دقيقة مشتركة.[2]

2.2.2. بروتوكول تخصيص المؤشر الديناميكي:

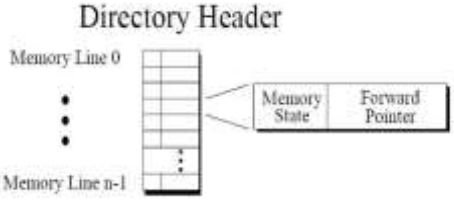
يحافظ بروتوكول تخصيص المؤشر الديناميكي على الدقة في مشاركة المعلومات حتى في الأجهزة ذات الأحجام الكبيرة جداً. كما في بروتوكول الـbit-vector كل عقدة في بروتوكول تخصيص المؤشر الديناميكي تحوي مدخلات الدليل (directory entry) لكل خط كاش للذاكرة الرئيسة في هذه العقدة. كما هو مبيّن في الشكل (9).

الشكل (9) بنى المعطيات لبروتوكول تخصيص المؤشر الديناميكي. معلومات التشارك توضع في قائمة مترابطة(linked list) يشار إليها من خلال مجموعة مؤشرات مخزنة في مخزن ال pointer/link.

تحوي مدخلات الدليل بت الحالة كما هو الأمر في بروتوكول الـbit vector، ولكن بدلاً من أن يكون هناك شعاع بتات للعقد المشتركة، مدخلات الدليل تخدم فقط كترويسة للدليل directory header، فضلاً عن معلومات إضافية مشتركة محفوظة في بنية link list. من أجل زيادة الفعالية، ترويسة الدليل تحوي بتاً محلياً ليدلّ على حالة الكاش للمعالج المحلي فضلاً عن حقل المشارك الأول في القائمة List. إنّها أيضاً تحوي مؤشراً يدلّ على باقي القائمة من المشاركين. المتبقي من قائمة المشاركة انتقلت إلى بنية معطيات ثابتة تسمى pointer/link storeالتي تحوي على مؤشر إلى مشارك آخر ورابط إلى العنصر التالي في قائمة المشاركة. في البداية مخزن الـ pointer/link يكون مربوطاً ببعضه بعضاً من خلال قائمة كبيرة فارغة. عندما يقرأ المعالج خط كاش، يزيل المتحكم مؤشراً جديداً من رأس القائمة الفارغة، ويربطها برأس الـlinked list المنشأة من ترويسة الدليل نفسه. إن ذلك مشابه تماماً لعمل بت الوجود في بروتوكول الـ bit vector. عندما يتم كتابة خط كاش يخترق المتحكم الـlinked list للمشاركين المحتفظة في ترويسة الدليل لذلك الخط، وترسل رسالة عدم صلاحية إلى المشتركين كلّهم في القائمة. عندما تصل إلى نهاية القائمة يكون قد تم استخدام كامل القائمة، وتوضع في القوائم الفارغة. يظهر بشكل واضح أن تنظيم بروتوكول تخصيص المؤشر الديناميكي لا يستغل الوقت بكفاءة عالية كما في التنظيم البسيط للـ bit vector. والتعقيد الثاني هو صعوبة تحقيق هذا التنظيم بشكل عتادي.[4]

3.2.2. بروتوكـــول الواجهـــة المتدرجـــة المتماسـكة (Scalable Coherent Interface):

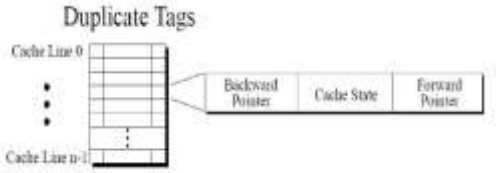
يعرف بروتوكول الواجهة المتدرجة المتماسكة SCI أيضاً بـIEEE Standard 1596-1992. هدف بروتوكول الـ-SCI هو أن يتناسب بمرونة مع عدد كبير من العقد مع تحميل زائد للذاكرة بأصغر قدر ممكن. الفكرة الأساسية من هذا البروتوكول هو إبقاء الـlinked list لمشاركين، ولكن على خلاف بروتوكول تخصيص المؤشر الديناميكي، هذه القائمة هي قائمة مزدوجة وموزعة عبر عقد الآلة. كما هو مبيّن في الشكل (10).



الشكل (10) ترويسة الدليل لبروتوكول الـSCI.

ترويسة الدليل في SCI هي أصغر منها في البروتوكول السابق لأنّها تحوي فقط على مؤشر للعقدة الأولى من قائمة المشاركة.

لاجتياز القائمة المشتركة، على البروتوكول أن يتبع المؤشر في ترويسة الدليل عبر الشبكة إلى أن تصل إلى المعالج المؤشر عليه. على هذا المعالج أن يحافظ على بنية duplicate set of tags معطيات تحاكي الحالة الحالية لذاكرة الكاش الخاصة به. الـ duplicate set of tags كما هي مبّينة في الشكل 11 تحوي مؤشر نحو backward pointer الخلف يحوي الحالة الحالية لذاكرة الكاش، وتحوي مؤشراً نحو الأمام forward pointer يؤشر على المعالج التالي في القائمة.



الشكل (11) بنية معطيات duplicate set of tags محفوظة بكل عقدة في بروتوكول الـSCI. توجد بطاقة مضاعفة واحدة لكل خط كاش في المعالج المحلي.

إنّ التطبيق الرسمي لبروتوكول الـSCI يتم بحفظ هذه البنية من المعطيات مباشرة في ذاكرة الكاش الثانوية للمعالج الرئيسي، وهكذا يكون بروتوكول الـSCI، وكأنّه بروتوكول معتمد على الكاش. في التطبيق العملي، ونظراً الى أنّ ذاكرة الكاش يجب أن تكون أصغر ما يمكن، وأسرع ما يمكن، وترتبط بعقدة وحيدة المعالج، وتكون تحت سيطرة المعالج بشكل قوي، معظم البنى التي تستخدم بروتوكول الـSCI تنفذ بنية المعطيات هذه من خلال بطاقة كاش مضاعفة في الذاكرة الرئيسة للنظام لكل عقدة.[1]

إن الطبيعة الموزعة لبروتوكول الـSCI لها حسنتان: 1- إنّها تقلل الحمل الزائد على الذاكرة إلى حد كبير بسبب الترويسات الأصغر للدليل وبسبب أنّ معلومات البطاقة المضاعفة تضيف فقط كمية صغيرة من الحمل الإضافي الى المعالج متناسب مع عدد خطوط كاش هذا المعالج بدلاً من العدد الأكبر الذي هو لخطوط كاش الذاكرة الرئيسة المحلية. 2- فهي تقلل الـ hot-spotting في نظام الذاكرة.

3. ما البروتوكول الأفضل؟

بعد مناقشة 3 بروتوكولات من بروتوكولات اتساق الكاش السؤال الرئيسي الذي يتبادر للذهن هو: ما البروتوكول الأفضل؟

في السابق كان جواب هذا السؤال يشكل صعوبة كبيرة للإجابة عنه. غالباً يتم إظهار ميزات كل بروتوكول بدقة

شديدة من قبل مناصري هذا البروتوكول. إنّ هذا السؤال قابل للنقاش لأنّه و بشكل عام عند تنفيذ أي بروتوكول اتساق كاش يكون مرتبطاً تماماً بآلة محددة، و بالعكس. نظراً الى أنّ أي نظام متعدد المعالجات المربوطة مع بعضها بشكل سلكي يملك بروتوكول اتساق الكاش، لذا يقارن بين البروتوكولات من حيث الأداء عبر آلات مختلفة. هذه المقاربة مدانة بسبب الاختلاف بين بنى الأجهزة أو الشك المحتم في المقارنة الذي يجعل الأمر صعباً جداً. لحسن الحظ، يوجد الآن حل لمشكلة صعوبة المقارنة بين خوارزميات اتساق الكاش. يتوضع الحل في مرونة البنية المتعددة المعالجاتStanford FLASH. باستخدام البنية FLASH المتعددة المعالجات كبنية للتجارب أصبح من الممكن وضع معايير رقمية فضلاً عن المعايير البحثية عن بروتوكولات اتساق الكاش وتساعد في الإجابة على الأسئلة الآتية:

- هل هناك بروتوكول وحيد مثالي للتطبيقات كلّها؟
- هل البروتوكول المثالي يتغير تبعاً لتغير حجم الآلة ؟
- هل البروتوكول المثالي يتغير بتغير حجم الكاش؟
- هل وجدت بنية وحيدة تنجز أداءَ قوياً مقابل مجال عريض من أحجام الآلات وخصائص التطبيقات؟

4. البروتوكولات البرمجية:

توجد بروتوكولات برمجية تعرض اتساقاً ببنى عتادية محدودة إمَّا تكون مبنية مع المترجم compiler، أو برمجيات حاضنة لهذا الغرض.

1.4. الحلول البرمجية:

يُقَسِّمُ المترجم المعطيات ويُفَرِّقُها كمعطيات قابلة للتخزين في الكاش ومعطيات غير قابلة. فقط المعلومات القابلة للقراءة فقط هي التي تعدّ معلومات قابلة للتخزين في الكاش، ويمكن وضعها في ذاكرة كاش خاصة. أمّا باقي المعلومات كلّها فهي التي ليست للقراءة فقط، ويمكن وضعها في ذاكرة كاش عامة إذا كان ذلك متاحاً.

2.4. تصنيف البروتوكولات البرمجية لاتساق الكاش:

توجد عدة معايير فصل لتصنيف البروتوكولات البرمجية موضّحة بالجدول (1).

3.4. الاتساق البرمجي مع دعم مادي محدود:

على المترجم أن يولد كوداً متسقاً تمامَ وكأنّه لا توجد آلية مادية من أجل الحفاظ على اتساق الكاش. يحوي الهاردوير في أثناء عمله فجوات زمنية تتجدّد عند كل عملية كتابة. عند القراءة، يخلق المترجم قراءات متسقة التي تقوم بالتحقق من البطاقة للتأكد من أن المعطيات متسقة. بالاعتماد على المترجم لاكتشاف القراءات التي يمكن أن تكون غير متناسقة، والعتاد يجب أن يحتفظ بهذه البطاقات.

الجدول (1) يوضّح معايير الفصل لتصنيف البروتوكولات البرمجية

نوع المعايير	الوصف
الديناميكي	تحليل زمن الترجمة و التنفيذ
الانتقائي	مستوى عمليات الاتساق
القيود	ذو قيود شديدة أو ذو قيود تعتمد على لحالة في الأداء
قابلية التكيف	يمكن للبروتوكول أن يتكيف مع أنماط الوصول
التفاصيل	حجم و بنية معلومات الاتساق وبنيتها
الحظر	كتلة البرنامج التي يكون فيها الاتساق حصرياً
تحديد المواقع	تحديد مواقع التعليمات التي يمكن أن يحدث عندها اتساق الكاش
التحديث	كيف تحدّث الذاكرة بعد الكتابة
الفحص	كيفية كشف الكتل التى لا تحدث فيها مشكلة اتساق الكاش

باستخدام هذه البطاقات، أيضاً من الممكن خلق كتل من عدم الصلاحية. توجد تقنيات عدّة تستخدم هذه البطاقات. إذا لم يكن لدى القسم العتادي بطاقات زمنية، طوّر بترسون ولي خوارزمية تستخدم فقط نقل الصفحات

كهاردوير وجدول الحالة. تبقى المعلومات المشتركة في يد البرمجيات في مستوى الصفحة. عند الوصول إلى الصفحة أو عند الإخفاق، تقوم البرمجيات بعملية فحص المعلومات المشتركة، وتعديل جدول الصفحات، ويقوم بإيجاد أفعال متسقة. إنّ تحقيق عملية الاتساق بشكل برمجي تكون ذات كلفة زمنية أكبر.

الخاتمة

تعطينا الدراسة السابقة ملخصاً عن أنواع بروتوكولات اتساق الكاش، وعن أهمية هذه البروتوكولات في الأنظمة متعددة المعالجات. وبالمقارنات السابقة نجد أنّ من الضروري استخدام بروتوكولات الدليل في الأنظمة ذات المتعددة المعالجات الكبيرة لكي تزيد من أدائها، في حين استخدام بروتوكولات الاستطلاع يؤدي إلى نتائج أفضل في الأنظمة ذات المتعددة المعالجات الصغيرة، وبروتوكول الـbit-vector يستخدم في الآلات قليلة ومتوسطة عدد المعالجات كونه سريعاً وفعالاً، ويبقى موضوع البروتوكول الأمثل قابلاً للنقاش لأنّ تنفيذ أي بروتوكول اتساق كاش يكون مرتبطاً تماماً بآلة محددة، ولكن يمكن الآخذ بالحسبان الدراسة السابقة لتحديد البروتوكول الأنسب.

REFERENCES

[1] Juan Gomez-Luna Herruzo and Jose Ignacio Benavides, MESI Cache Coherence Simulator for Teaching Purposes. CLEI ELECTRONIC journal pp 1-7, 2009.

[2] John L. Hennessy, David A. Patterson, David Goldberg, Computer Architecture: A quantitative Approach, fourth edition, P579, 2007

[3] Yong J. Jang and Won W.R. Evaluation of Cache Coherence Protocols On Multi-Core Systems with Linear Workloads, ISECS International Colloquium on Computing, Communication, Control, and Management, pp 1-4, 2009.

[4] M. Heinrich, J. Hennessy, and A. Gupta, the Performance and Scalability of Distributed Shared Memory Cache Coherence Protocols, IEEE Transactions on Computers, pp 1- 7, 1999.

Received	2017/12/05	إيداع البحث
Accepted for Publ.	2018/11/28	قبول البحث للنشر