SCORE یک روش جایگزینی کش بر اساس امتیاز دهی به هر بلوک

محمد صادق ریاضی، سیدمحمد جواد سید طالبی

دانشکده برق

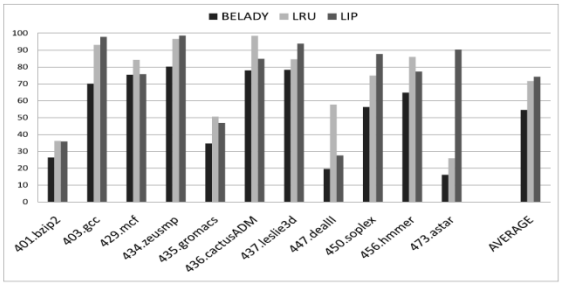
دانشگاه صنعتی شریف

چکیده

در این مقاله ما به بیان و بررسی یک روش جدید (SCORE) برای جایگزینی خط‌های داده در کش می‌پردازیم که در آن از روش امتیازدهی استفاده می‌شود[1]. در مقاله [1] ادعا شده است که نتایجی که با توجه به محک[1]های CPU2006 SPEC بدست آمده است، نشان می‌دهند که میانگین IPC، ۴,۹٪ نسبت به روش LRU و ۷,۴٪ نسبت به LIP افزایش پیدا کرده است. هدف ما در این مقاله بررسی و تحلیل این روش جایگزینی و احراز صحت ادعای مطرح شده در [1] پیرامون بهبود IPC در این روش است.

۱. مقدمه

تحقیقات پیشین نشان داده‌اندکه روش‌های جایگزینی داده در کش اثر به سزایی را در راندمان سیستم‌ها دارند. متاسفانه، روش‌های قابل پیاده‌سازی مانند LRU، MRU و LIP در برخی از کاربردها به خوبی و در برخی دیگر ضعیف کار می‌کنند. این نتایج را می‌توانید در شکل ۱ که miss rate لایه آخر کش (LLC) برای زیرمجموعه از محک‌های SPEC CPU2006 نشان داده است، مشاهده نمایید.



شکل ۱. نمودار Missهای روش‌های مختلف

در ادامه به مقایسه الگوریتم Belady و LRU وLIP می‌پردازیم. تفاوت اصلی بین LRU و LIP در موقعیت اولیه خط جدید کش در stack است. در روش LRU خط جدید در موقعیت MRU قرار می‌گیرد، در حالی که در LIP خط جدید در موقعیت LRU قرار می‌گیرد. بعد از یک hit، هر دو روش خط hit شده را در موقعیت MRU قرار می‌دهند. نتایج نشان می‌دهند که قرار دادن خط جدید در موقعیت MRU برای کاربردهای با محلیت بالا بهتر است؛ در حالیکه قرار دادن خط جدید در LRU برای کاربردهایی بهتر است که در اغلب موارد فقط یک بار به خط‌های کش دسترسی پیدا می‌کنند. روش‌های پویای توضیح داده شده در [12] به طور پویا روشی را مابین LRU و MRU انتخاب می‌کنند که

[1] Benchmarks

راندمان را افزایش دهد. ولی همچنان فضا برای پیشرفت الگوریتم Belady وجود دارد. این مقاله روش را بر مبنای امتیاز دهی پیشنهاد می‌کند که از امتیازدهی برای انتخاب خط victim استفاده می‌کند. به هر خط از کش یک امتیاز نسبت داده می‌شود که بیان کننده چگونگی دسترسی به آن خط در مقایسه با خطهای دیگر در آن گروه کش است. به هر کدام از این خطوط هنگام انتقال آنها از حافظه به کش یک امتیاز اولیه نسبت داده می‌شود. در صورت یک hit برای خط، امتیاز آن خط افزایش پیدا می‌کند و در غیر این صورت کاهش پیدا می‌کند. این امتیازها در حالت کلی نشانگر رفتار آن خط در آینده است: خطوط با امتیاز کمتر، احتمال کمتری نسبت به خطوط با امتیاز بیشتر برای دسترسی مجدد دارند.

روش‌های دیگر مانند LRU، MRU و LIP می‌توانند بر اساس همین سیستم امتیاز دهی با تغییر پارامترهای آن پیاده‌سازی شوند. این روش همچنین می‌تواند با تغییر پویای پارامترهای خود در طول اجرای برنامه با آن سازگاری بیشتری پیدا کند. نتایج نشان می‌دهد که این روش نسبت به روش‌های دیگر جایگزینی نرخ miss کمتری دارد؛ همچنین این روش پایدارتر بوده و انحراف معیار کمتری دارد که سرعت بیشتر و نرخ miss کمتری را نتیجه می‌دهد.

در این مقاله رفتار این روش را در لایه آخر کش بررسی شده است. این کش دارای ۱۶ راه (وی) و 1MB حافظه است. توجه به این نکته حایز اهمیت است که این روش را می‌توان در لایه‌های دیگر کش نیز به کار گرفت.

۲ روش جایگزینی امتیازی (SCORE)

در این روش به هر خط کش امتیازی نسبت داده می‌شود که نشانگر رفتار آن خط در گذشته است. خطهای با امتیاز کم سوژه‌های خوبی برای جایگزین شدن هستند. در یک سمت، امتیازها به صورت پویا محاسبه می‌شوند و در سمت دیگر امتیازها برای جایگزینی خطهای کش مورد استفاده قرار می‌گیرند.

در سیستم طراحی شده، صفر به عنوان کمترین امتیاز لحاظ شده و اگر n را تعداد بیت مورد نظر برای امتیاز دهی در نظر بگیریم، خواهیم داشت:

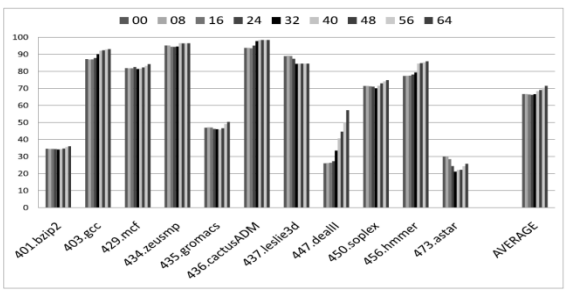
$$SCORE_{MAX} = 2^n - 1$$

افزایش تعداد بیت‌های در نظر گرفته شده به افزایش ریزدانگی و در نتیجه افزایش دقت این روش می‌انجامد. در اینجا ما به هر خط یک عدد یکتا نسبت می‌دهیم و از آنجا که ۱۶ راه (Way) داریم، نیاز به ۴ بیت برای این امر وجود دارد. البته در روش SCORE می‌توان تعداد بیت‌ها را در زمان طراحی کش تغییر داد.

۱,۲ تعیین مقدار اولیه امتیازها

بعد از آنکه یک خط از حافظه به کش انتقال پیدا می‌کند، به آن یک *امتیاز اولیه* نسبت داده می‌شود. خطوطی که در

آینده نزدیک به آنها دسترسی پیدا می‌شود، باید امتیاز بالایی به آنها نسبت داده شود تا به مدت کافی در کش نگه داشته شوند. این امر در کاربردهای با محلیت بالا خود را نشان می‌دهد. همچنین خطوطی که در آینده نزدیک به آنها دسترسی نمی‌شود، می‌بایست امتیاز کمتری به آنها نسبت داده شود تا در آینده به عنوان سوژه قرار گرفته و جایگزین خطوط دیگر شوند. که این امر خود را در کاربردهای با محلیت پایین مثل کاربردهای Pointer Intensive خود را نشان می‌دهد. مقدار خاصی که به عنوان امتیاز اولیه تخصیص داده می‌شود نه تنها باید وابسته به برنامه باشد، بلکه باید به مراحل مختلف اجرا نیز وابسته باشند. در بهترین شرایط برای هر فاز اجرای برنامه باید مشخص شود کدام خط از کش دوباره استفاده می‌شود تا به آن امتیاز اولیه بالایی داده شود و کدام خط از کش دوباره استفاده نمی‌شود تا به آن امتیاز اولیه پایینی نسبت داده شود. جالب است بدانید که بهترین امتیاز برای یک کاربرد همواره مقدار کمینه یا بیشینه امتیازها نیستند (بر خلاف LRU و LIP)، بلکه جایی بین این دو مقدار است. این امر به این خاطر است که برنامه‌های رایج حاوی هر دو نوع دسترسی با محلیت بالا و پایین هستند. با تحلیل امتیاز اولیه‌های مختلف می‌توان به بهترین امتیاز اولیه هر برنامه پی برد.



شکل ۲. اثر مقدار اولیه امتیازها بر نرخ Miss

نتایج نشان می‌دهند که برای یک کاربرد خاص، بهترین مقدار اولیه می‌تواند هر مقداری بین صفر و بیشینه امتیاز باشد. این بدان معنا است که امتیاز اولیه به خود آن برنامه و فازهای مختلف اجرایی آن بستگی دارد، بنابراین می‌بایست امتیاز اولیه به صورت پویا اختصاص داد. این مقادیر اولیه پویا با مشاهده تعداد missها در یک تعداد مشخص دسترسی به کش (یک دوره) مشخص می‌شوند. به این ترتیب که ما یک جهش در نظر می‌گیریم که با توجه به مسیر حرکت به امتیاز اولیه اضافه و یا کم می‌شود. مسیر حرکت این گونه تعیین می‌شود که اگر در دوره حاضر تعداد missها کمتر از دوره قبل بود مسیر حرکت تغییر نمی‌کند و در غیر این صورت مسیر حرکت تغییر خواهد کرد.

۲,۲ تغییر امتیازها

یک مشخصه دیگر روش جایگزینی مبتنی بر امتیازدهی (SCORE) تغییر امتیازها بر مبنای دسترسی به کش است. اگر یک خط hit شود، امتیاز آن با یک عدد به نام *سرعت افزایش* جمع می‌شود. اگر یک خط دیگر در همان گروه hit شود. ما همچنین یک *سرعت کاهش* تعریف می‌کنیم که در واقع سرعت کاهش امتیازها است، زمانی که خط جایگزین شود و یا خط دیگری در آن گروه hit شود. هنگامی‌که یک خط جدید از حافظه آورده می‌شود یا در کش hit می‌شود، تمام خطهای دیگر در آن گروه امتیازشان کاهش پیدا می‌کند. به همین علت برای جلوگیری از کم شدن بی رویه

امتیازها سرعت افزایش را از سرعت کاهش بیشتر قرار می‌دهیم. امتیاز اولیه و تغییر سرعت‌ها دو مشخصه مهم روش مبتنی بر امتیاز هستند. یک مشخصه دیگر که به همه اعضای گروه اعمال می‌شود، مجموع امتیازات همه خط‌های موجود در یک گروه است. امتیاز اولیه و سرعت تغییرات به گونه ای انتخاب می‌شود که امتیاز یک گروه خیلی بزرگ و یا خیلی کوچک نشود.

۳,۲ انتخاب خط سوژه

با اینکه به نظر می‌رسد خط با پایین ترین امتیاز، بهترین گزینه برای جایگزینی است اما تمام خطوطی که امتیاز پایینی دارند می‌توانند سوژه‌های خوبی برای جایگزینی باشند. به همین خاطر در SCORE یک *امتیاز آستانه* در نظر گرفته شده است که برای خط‌های با امتیاز کمتر از این آستانه از روش رندوم (Random) استفاده می‌شود و در حالتی که هیچ خطی امتیاز کمتر از آستانه نداشته باشد خط با کمترین امتیاز سوژه جایگزینی می‌شود.

۳ روش انجام کار

در مقاله اصلی، فایل‌های زیر از SPEC CPU 2006 تحت

```
401.bzip2, 403.gcc,
429.mcf,434.zeusmp, 435.gromacs,
436.cactusADM, 437.leslie3d,
447.dealII, 450.soplex, 456.hmmer,
473.astar.
```

کامپایلرهای GNU در سیستم عامل Linux کامپایل شده‌اند:

و توسط شبیه‌سازی Pin، فایل Trace آنها ساخته شده است. این فایل‌های Trace برای ۵۰۰ میلیون و ۵ میلیارد دستور بعد از پرش از ۴۰ میلیارد دستور جمع آوری شده است و توسط نرم افزار CMP$im شبیه‌سازی شده است. شبیه‌سازی بر روی یک مدل با کش سه لایه و یک هسته ساده با اجرای خارج از ترتیب اجرا شده است.

گروه ما در ابتدا سعی داشت پس از آشنایی با نرم افزار Simple Scalar شبیه‌سازی‌های مربوط به این مقاله را توسط آن انجام دهد اما در ادامه با آشنایی با نرم افزار CMP$im دریافتیم از آنجایی که این شبیه ساز مختص شبیه‌سازی کش و روش‌های جایگزینی بلوک‌های کش طراحی شده است به این نتیجه رسیدیم که انجام شبیه‌سازی با همین نرم افزار مناسب تر است.

نرم افزار CMP$im یک نرم افزار Open Source است که کدهای آن و نحوه نصب آن در [13] توضیح داده شده است. برای شبیه‌سازی الگوریتم جایگزینی دلخواه در این نرم افزار کافیست فایل‌های replacement_state.cpp و replacement_state.h که در پوشه src محل نصب نرم افزار موجود است را با توجه به الگوریتم دلخواه خودمان تغییر دهیم و سپس تمام فایل‌ها را توسط make file و کامپایلر gcc کامپایل کنیم. Source file مربوط به پیاده سازی الگوریتم SCORE ضمیمه شده است. سپس می‌بایست از

روی یک فایل باینری Trace مورد نظر برای شبیه‌سازی ساخته شود. دستور زیر نحوه ساختن فایل Trace برای یک میلیون دستور العمل بعد از پرش از یک میلیون دستورالعمل را برای باینری فایل ls در دایرکتوری bin سیستم عامل Linux نشان می‌دهد:

```
pinkit/pin-2.7-31933-gcc.3.4.6-
ia32_intel64-linux/pin -t
./bin/CMPsim.gentrace.32 -threads 1
-fwd 1 -icount 1 -o traces/lsal.out
-- /bin/ls -al
```

این دستور یک فایل Trace به عنوان خروجی تولید می‌کند که باید به شبیه ساز داده شود تا آن را توسط یک کش تعریف شده و الگوریتم جایگزینی خاص شبیه‌سازی کند. این امر توسط قطعه کد زیر انجام می‌شود:

```
../bin/CMPsim.usetrace.32 -threads
1 -t ../traces/ls.out.trace.gz -o
ls.stats -cache UL3:1024:64:16 -
LLCrepl 2
```

در کد بالا یک کش با سایز ۱۰۲۴ کیلوبایت، سایزهای خط ۶۴ کیلوبایتی و تعداد راه‌های ۱۶ تعریف شده است که فایل Trace از قبل ایجاد شده را با یکی از الگوریتم‌های LRU، Random و یا الگوریتم دلخواه ما شبیه‌سازی می‌کند. عدد ۲ در انتهای کد به معنی شبیه‌سازی با الگوریتم داده شده ما است. نتایج این شبیه‌سازی در قسمت بعد آورده شده است.

بررسی و نتایج

نتایج اجرای کدهای بالا را در زیر می‌توان مشاهده کرد:

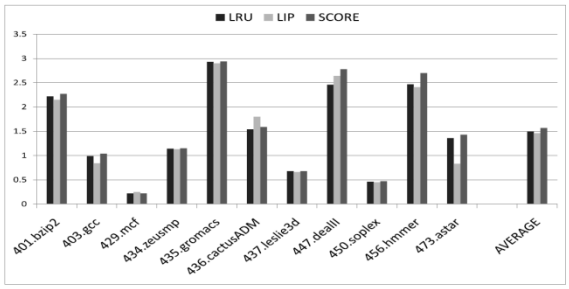
```
[CMP$im] Finished Requested Instruction Count 1163935
for App 0 at Time: 739470

Full Run Summary: (Global Instructions: 1163935)
Thread ID: 0 ICOUNT: 1163935 CYC: 739471 CPI:
0.63532 Global Cycles: 739471 LLC Misses: 2169

Region of Interest Summary:
Thread ID: 0 ICOUNT: 1163935 CYC: 739470 CPI:
0.635319 Global Cycles: 739471 LLC Misses: 2169
```

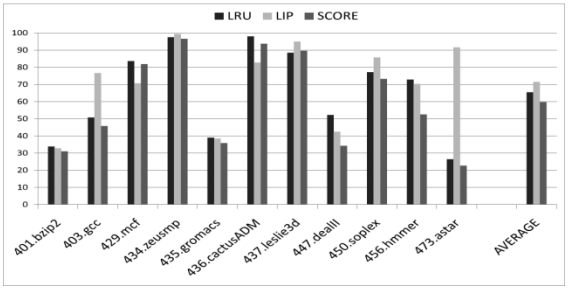
البته به دلیل حجم کم کد اجرا شده در برنامه ls نتایج بدست آمده یعنی تعداد missهای کل و CPI بدست آمده در این روش تفاوت چشم گیری با روش LRU ندارد. تنها گام باقی مانده شبیه‌سازی الگوریتم مورد نظر با محک‌های داده شده در مقاله و مقایسه آن‌ها با نتایج اعلام شده در مقاله است. به دلیل Open Source نبودن این محک‌ها، بعد از جستجوی فراوان در اینترنت توانستیم کد باینری بعضی از این محک‌ها را پیدا کنیم که پیوست نیز شده است. اما بعد از شبیه‌سازی نتایج بدست آمده با نتایج موجود در مقاله همخوانی لازم را نداشت. البته این می‌تواند به خاطر این باشد که ما Source file محک‌ها را در اختیار نداشتیم تا آنرا مشابه روش گفته شده در مقاله کامپایل کنیم و این تفاوت در نتایج می‌تواند به دلیل تفاوت در فایل ورودی شبیه ساز باشد. در ادامه به بیان نتایج مطرح شده در مقاله می‌پردازیم.

شکل ۳ IPC را برای سه روش مختلف جایگزینی محک‌های مختلف نشان می‌دهد.



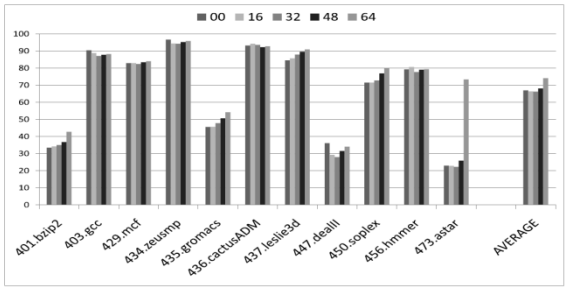
شکل ۳. IPC برای ۳ روش متفاوت جایگزینی

و شکل ۴ تفاوت miss rate در این سه روش را نشان می‌دهد.



شکل ۴. تفاوت miss rate در ۳ روش مختلف

همچنین می‌توان تاثیر میزان آستانه را در miss rate در شکل ۵ مشاهده کرد.



شکل ۵. اثر آستانه بر miss rate

در نهایت ادعای مقاله اصلی این است که روش SCORE، IPC را به طور میانگین به میزان ۴,۹٪ نسبت به LRU و ۷,۴٪ نسبت به LIP بهبود می‌دهد.

مراجع

[1] N. Duong, R. Cammarota, D. Zhao, T. Kim and A. Veidenbaum. SCORE: A Score-Based Memory Cache Replacement Policy. JWAC-2: Championship Branch Prediction Program 2011.

[2] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.*, 5(2):78{101, 1966.

[3] R. B. Gramacy, M. K.Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In *NIPS*, pages 1465{ 1472, 2002.

[4] D. Grund and J. Reineke. Estimating the performance of cache replacement policies. In *MEMOCODE '08: Proceedings of the 6th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, pages 101{ 111, June 2008.

[5] F. Guo and Y. Solihin. An analytical model for cache replacement policy performance. *SIGMETRICS Perform. Eval. Rev.*, 34(1):228{239, 2006.

[6] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1{17, 2006.

[7] A. Jaleel, R. S. Cohn, C. keung Luk, and B. Jacob. Cmp$im: A binary instrumentation approach to modeling memory behavior of workloads on cmps. Technical report, 2006.

[8] J. Jeong and M. Dubois. Cache replacement algorithms with nonuniform miss costs. *IEEE Trans. Comput.*, 55(4):353{365, 2006.

[9] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Comput.*, 50(12):1352{1361, 2001.

[10] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 190{200, New York, NY, USA, 2005. ACM.

[11] E. J. O'neil, P. E. O'Neil, and G. Weikum. The lru-k page replacement algorithm for database disk bu®ering,1993.

[12] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 381{391, New York, NY, USA, 2007. ACM.

[13]http://www.jilp.org/jwac-1/replacement_framework_description.html