گزارش پروژه میان‌ترم معماری کامپیوتر

شبیه سازی حافظه‌های نهان خاص

دانشگاه صنعتی امیرکبیر، دانشکده مهندسی کامپیوتر و فناوری اطلاعات

پارسا اسکندرنژاد - امیرعلی سجادی

فهرست مطالب

I. مقدمه

سرعت پردازنده‌ها طی سال‌های اخیر رشد چشم گیری داشته است در حالی که افزایش سرعت حافظه ، با سرعت کمی‌در جریان است. به علت این که حافظه و CPU با هم کار می‌کنند سرعت کلی عملیات را حافظه تعیین می‌کند. مشکل دیوار حافظه، به این اشکال گفته می‌شود که هر چقدر هم سرعت پردازنده بالا برود، به دلیل این که به درخواست‌های مکرر از حافظه نیاز دارد، سرعت کلی اجرا توسط حافظه محدود می‌شود.

هدف از استفاده از حافظه‌ی نهان[1] این است که درخواست‌هایی که احتمال صدا شدن آن‌ها بیشتر است به پردازنده نزدیک‌تر باشند تا سرعت دسترسی به آن‌ها بیشتر شود.

برای این منظور حافظه‌ی نهان از اصل مجاورت زمانی و مکانی بین آدرس‌های درخواستی بهره می‌برد که با توجه به ماهیت برنامه‌های فعلی کامپیوتری و نوع کاربری آنها برقرار است؛ یعنی اگر یک آدرس درخواست شود احتمالش زیاد است که دوباره همان آدرس درخواست شود یا وقتی یک آدرس درخواست شود، احتمالش بالا است که آدرس‌های مجاور آن درخواست شوند (مثل دستورات یک حلقه‌ی تکرار در کد یک برنامه که وقتی دستوری از حافظه خوانده شود دستورات بعدی آن بعد از آن درخواست می‌شوند و برای تکرارهای بعدی دستورات حلقه همان دستورات دوباره از حافظه خواسته می‌شوند.) با توجه به این اصل، زمانی که یک داده درخواست شد، به همراه حافظه‌های مجاور در حافظه نهان قرار می‌گیرد تا سرعت دسترسی به آن‌ها بیشتر شود.

II. هدف

اضافه شدن حافظه‌ی نهان ، سرعت درخواست از حافظه را تا حد قابل توجهی بالا برد اما در مقابل سرعت پردازنده‌ها هنوز این سرعت ناچیز بود و هنوز فاصله‌ی زیادی بین سرعت حافظه و پردازنده حس می‌شد.

بررسی‌ها نشان می‌دهد که استفاده از نوعی حافظه نهان کوچکتر ولی سریعتر از حافظه نهان اصلی در بین حافظه اصلی و حافظه نهان اصلی، میزان نرخ موفقیت را بهبود می‌بخشد.

از این نوع حافظه‌های نهان می‌توان به «Victim cache» و «Trace cache» اشاره کرد.

[1] Cache

1. Victim cache

اگر حافظه‌ی نهان اصلی، از روش نگاشت مستقیم[2] استفاده کند دسترسی به اطلاعات سریع تر است ولی تعداد conflictها افزایش می‌یابد و ممکن است با وجود فضای خالی در قسمت‌های دیگر حافظه‌ی نهان ، در یک بلوک حافظه نهان به طور مداوم، بلوک آدرس جدید درخواست شده، جایگزین قبلی شود در حالی که احتمال دارد بلوکی که دفعه‌ی قبل درخواست شده بود، طبق اصل هم جواری زمانی و مکانی دوباره درخواست شود ولی به علت این که در حافظه‌ی نهان موجود نیست کارایی حافظه‌ی نهان کاهش می‌یابد.

می‌توان برای حل این مشکل از حافظه‌های نهان مجموعه انجمنی[3] و تمام انجمنی[4] استفاده کرد که در آن صورت سرعت دسترسی نگاشت مستقیم را نداریم. اما با استفاده از Victim cache هم سرعت دسترسی نگاشت مستقیم و هم تعداد کم conflict نگاشت تمام انجمنی را خواهیم داشت. Victim cache یک حافظه‌ی نهان با نگاشت تمام انجمنی با اندازه‌ی کوچک است که پس از آن که درخواست آدرس از حافظه‌ی نهان اصلی miss شد، دیگر به حافظه‌ی اصلی رجوع نمی‌شود و داده مورد نظر در صورت وجود، توسط Victim cache جا به جا می‌شود. و هنگامی که داده‌ای در حافظه‌ی نهان اصلی قرار می‌گیرد اگر conflict ای وجود داشته باشد، داده‌ای که قبلا در آن بلوک حافظه بوده مثل قبل دور ریخته نمی‌شود بلکه وارد Victim cache می‌شود و اگر Victim cache پر بود مثلا با سیاست LRU جایگزین می‌شود.

2. Trace cache

همچنین Trace cache نیز یک حافظه‌ی نهان خاص است که جریانی از دستورات را به صورت پویا ذخیره می‌کند که به آن Trace می‌گویند. این کار باعث افزایش پهنای باند استخراج و دریافت[5] دستورات می‌شود؛ دستوراتی که قبلا دریافت و decode شده اند در Trace cache ذخیره می‌شوند و وقتی دوباره درخواست شوند، نیاز به دریافت و decode دوباره نیست.

Traceها با حداکثر تعداد دستورات و حداکثر دستورات پایه‌ای مشخص می‌شوند؛ دو Trace می‌توانند دستور شروع یکسانی داشته باشند اما باید تعداد دستورات پایه‌ای‌شان متفاوت باشد. وقتی

[2] Direct mapping

[3] Set Associative

[4] Fully Associative

[5] fetch

پردازنده قصد اجرای دو دستور نامتوالی مثلا به دلیل یک jump را دارد اگر قبلا این دستورات را دریافت و decode کرده باشد دیگر نیازی به دانستن ماهیت دستورها در مدار حافظه نیست.

هدف این پروژه شبیه‌سازی Victim cache می‌باشد.

III. پیاده‌سازی

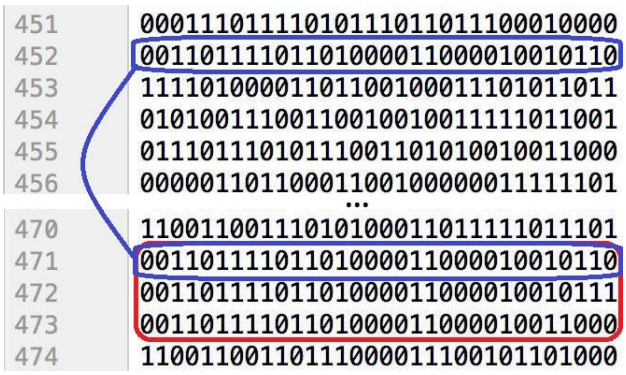
در فاز اول پروژه ابتدا باید فایل‌های ورودی 500 تایی ساخته شود. برای این کار ابتدا 100 آدرس 32 بیتی رندوم و به صورت رشته‌های باینری تولید می‌شود و سپس در 400 آدرس بعدی با تکرار این آدرس‌ها همجواری زمانی و همچنین با آوردن آدرس‌های مجاور، همجواری مکانی تولید می‌شود. در این الگوریتم برای ایجاد همجواری زمانی به احتمالات مختلف مثلا 20% حالات، همجواری‌های مکانی بین 1 تا 3 تولید می‌شود.

همچنین هر چه که به پایان این 500 آدرس نزدیکتر می‌شویم همجواری مکانی بیشتر می‌شود، به عنوان مثال در 100 آدرس آخر فقط 50 آدرس اول از 100 آدرس اول به طور تصادفی تکرار می‌شوند.

شکل کلی فایل تولید شده توسط این الگوریتم به صورت زیر است:

0-100	به صورت تصادفی از 0 تا $2^{32}-1$
100-250	تکرار به صورت تصادفی از 100 آدرس اول
250-500	تکرار به صورت تصادفی از 50 آدرس اول

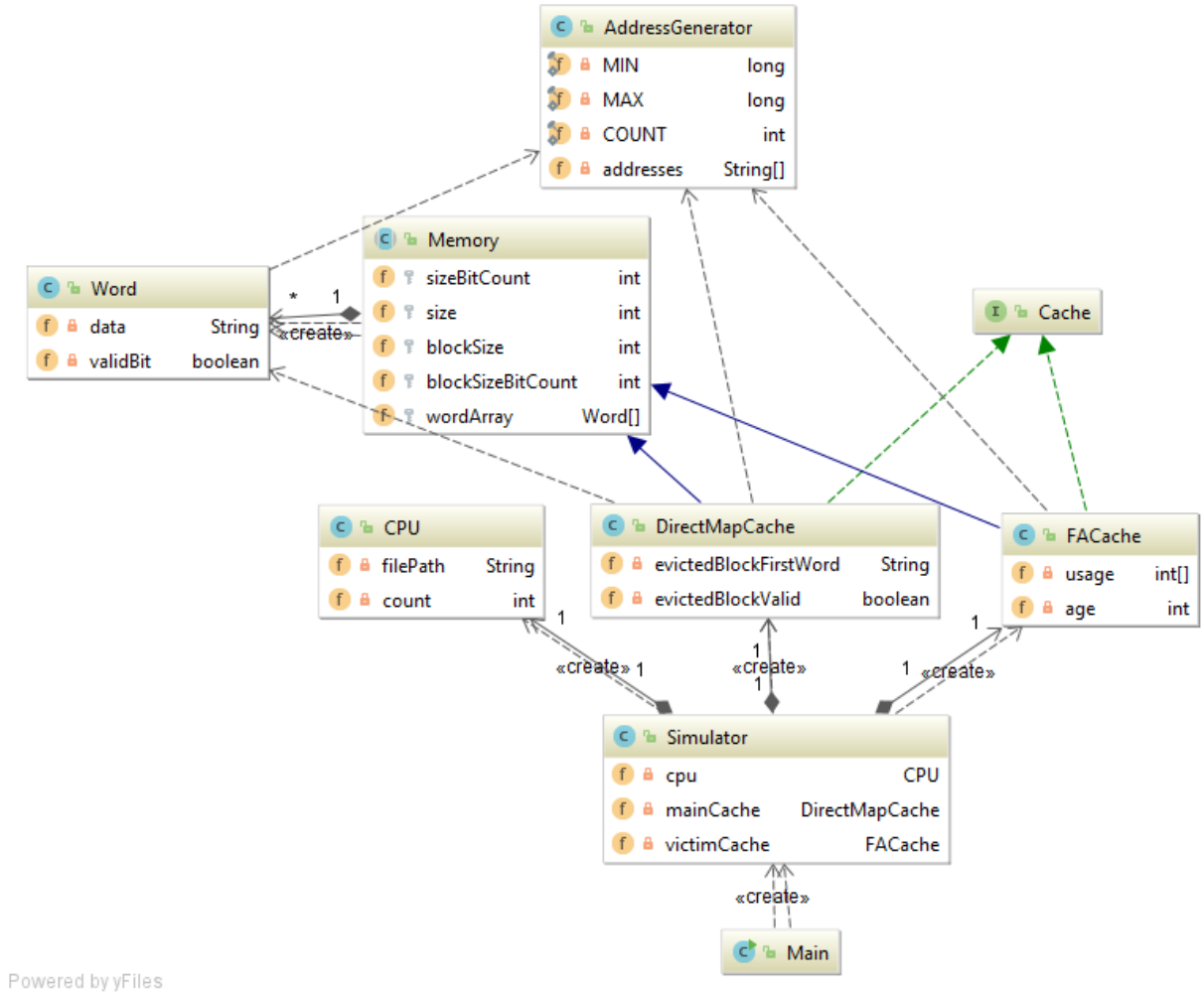
هر بار به احتمال 1/5 همجواری مکانی به مقدار 1 تا 3 تولید می‌شود.



به عنوان مثال در شکل بالا مستطیل قرمز همجواری مکانی و مستطیل آبی همجواری زمانی را نشان می‌دهد.

فاز بعدی پروژه شبیه سازی سیستم مورد نظر است.

دیاگرام زیر نحوه کلاس‌بندی‌ها را نشان می‌دهد:



توضیحات مختصر هر کلاس به این شرح است:

نام	توضیح
Word.java	نمایش دهنده یک آدرس 32 بیتی به همراه یک بیت valid
AddressGenerator.java	وظیفه ساخت آدرس را بر عهده دارد. همچنین دو تابع استاتیک تبدیل رشته بیت باینری به long و برعکس آن به وفور استفاده می‌شود.
CPU.java	فایل دستورات را به صورت یک لیست ارائه می‌دهد.
DirectMapCache.java	یک حافظه نهان نگاشت مستقیم را نمایش می‌دهد.
FACache.java	یک حافظه نهان تمام انجمنی را نمایش می‌دهد.
Simulator.java	CPU و حافظه‌ها را به هم وصل می‌کند و عملیات خواسته شده را انجام می‌دهد.

برای محاسبه نرخ موفقیت سه حالت ممکن است پیش بیاید:

1- **داده اصلی در حافظه نهان نگاشت مستقیم باشد.**

کاری که باید انجام بدهیم: هیچ

2- **داده اصلی در حافظه نهان نگاشت مستقیم نباشد ولی در Victim cache باشد.**

کاری که باید انجام بدهیم: داده مورد نظر با داده‌ای که در جای آن در حافظه نهان نگاشت مستقیم است جا به جا شود. (توجه شود که فقط جا به جایی باید صورت بگیرد و نباید از سیاست جایگزینی استفاده شود.)

3- **داده اصلی در هیچ کدام نباشد.** Miss!

کاری که باید انجام بدهیم: ابتدا داده را از مرحله بعد به حافظه نهان نگاشت مستقیم می‌آوریم و سپس داده رانده شده را طبق سیاست Victim cache (در اینجا LRU) به آن اضافه می‌کنیم.

۷. نتایج

برای مشاهده نتایج ابتدا 5 فایل ورودی را طبق الگوریتم گفته شده تولید کرده و آن‌ها را اجرا می‌کنیم.

اسم فایل	بدون استفاده از Victim cache	با استفاده از Victim cache
input1.txt	71%	75.8%
input2.txt	69.4%	76.4%
input3.txt	74.6%	77%
input4.txt	73.8%	76.2%
input5.txt	71.6%	76.6%
میانگین	72.08%	76.4%

به طور متوسط در حدود 4.32% افزایش کارایی داشتیم. لازم به ذکر است این مقدار، مقدار میانگین می‌باشد و در حالات دیگر حتی تا 12% افزایش موفقیت نیز مشاهده شد.

مشاهده شد که استفاده از Victim cache میزان نرخ موفقیت را افزایش داد.

۷. کارهای آینده

برای به دست آوردن دقت بیشتر می‌توانیم از فایل‌های واقعی ورودی استفاده کنیم. همچنین می‌توان تعداد و دقت فایل‌های ساخته شده را بیشتر کرد.

۷۱. منابع

1. http://istc-bigdata.org/index.php/memory-wall-what-memory-wall/
2. https://en.wikipedia.org/wiki/Trace_Cache
3. https://en.wikipedia.org/wiki/Victim_cache
4. http://www.ecs.umass.edu/ece/koren/architecture/VCache/home.html
5. **Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching**, Eric Rotenberg Computer Science Dept. Univ. of Wisconsin – Madison, Steve Bennett Intel Corporation, James E. Smith Dept. of Elec. and Comp. Engr. Univ. of Wisconsin - Madison