دانلود مقاله ماشينهاي همزماني

جهت مشاهده دانلود مقاله ماشينهای همزمانی به پايين همين صفحه مراجعه نماييد

تعداد صفحات : 12 صفحه

برای دریافت اینجا کلیک کنید

فرمت WORD قابل ویرایش

ماشينهاي همزماني

– همزماني

ماشينهاي همزماني با روالهاي نرم افزاري در سطح كاربر ساخته شده اند كه آن استنادي است كه دستورات همزماني موجود در سخت افزار.

براي چند پردازنده هاي كوچكتر يا وضعيت رقابتي پايينتر،قابليت كليد سخت افزاري در يك دستور بي‌وفقه يا ترتيب و توالي دستور در بازيابي ذره وار(اتميك) و تغيير يك مقدار است و مكانيزم همزماني نرم افزاري اين توانايي را مي سازد در اين بخش ما روي پياده سازي عمليات همزماني،باز كردن و قفل كردن تمركز مي كنيم. Locl وunlock مي توانند بطور مستقيم در يك ممانعت متقابل بكار روند،همچنين در بكار بردن مكانيزمهاي همزماني پيچيده تر.

در يك مقياس بزرگتر در چند پردازنده ها يا در وضعيت رقابتي بالاتر،همزماني كارائي بيشتري را دارد چون رقابتهاي بيشتر تأخيرهاي اضافي را بوجود مي آورد ما در اينجا بحث مي كنيم كه چگونه مكانيزمهاي همزماني اوليه روي تعداد،بيشتري از پردازنده گسترش مي يابد.

اسانس سخت افزار اوليه

در قابليت كليد ما مستلزميم همزماني را در يك چند پردازنده كه مجموعه اي از سخت افزارهاي اوليه با قابليت خواندن ذره وار و يك مكان يابي حافظه است را اجرا كنيم بدون چنين قابليتي هزينه ساخت همزماني اوليه خيلي بيشتر خواهد بود و تعداد پردازنده ها افزايش خواهد يافت تعدادي قاعدة دستورسازي براي سخت افزار اوليه وجود دارد كه در جهت بهبود قابليت خواندن ذره وار و مكان يابي مناسب استفاده مي شود و با چند راه مي توان خواندن و نوشتن ذره وار را بيان كرد. اين سخت افزار اوليه اساس ساخت بلوكهايي است كه در انواعي از عمليات همزماني سطح كاربر استفاده مي شود و همچنين شامل قفلها و مانع هاست.

بطور كلي در اين معماري نمي توان انتظار داشت كه كاربران روي سخت افزار اوليه كار كنند اما در عوض انتظار مي رود كه از سيستمهاي برنامه نويسي براي ساخت يك كتابخانه همزماني استفاده شود كه معمولاً يك پردازش پيچيده است.

حال بحث را با يك سخت افزار اوليه و چگونگي عمليات همزماني براي آن شروع مي كنيم يكي از انواع عمليات همزماني مبادله اتمي (atomic exchanye) است كه ارزش يك رجيستر را با حافظه عوض مي كند حال ببينيم چگونه از اين عمليات همزماني استفاده كنيم. فرض مي كنيم كه مي خواهيم يك قفل ساده بسازيم و در آن با ارزش ۰صفر نشان مي دهيم كه

قفل آزاد است و با ۱ نشان مي دهيم كه غير قابل استفاده است در رجيستر و حافظه آدرس مطابق قفل است دستور emchanye 1 را برمي گرداند اگر پردازنده قبلاً دستيابي شده و در غير اينصورت ۵ را برمي گرداند. در حالت ديگر آن مقدار با ۱ تغيير مي كند و با حصول۰ صفر از هر تغييري جلوگيري مي كند. بطور مثال فرض مي كنيم دو پردازنده داريم كه هر يك تلاش مي كند همزماني را عوض كند اين رقابت وقتي تمام مي شود . كه يكي از پردازنده ها تغيير را انجام مي دهد و ۰ را برگرداند و در اينصورت پردازنده دوم ۱ را باز خواهد گرداند آن كليد از مبادله اوليه براي اجدا كردن همزماني در عمليات اتميك استفاده مي كند. آن مبادله غيرقابل تقسيم است و دو مبادله همزمان با نوشتن مكانيزمهاي پشت سرهم (سريالي) مرتب مي شود.

تعداد ديگر از اتميك هاي اوليه وجود دارد كه در انجام همزماني بكار برده مي شود و همه آنها قابليت خواندن و update كردن حافظه دارند و همچنين وضعيتي كه مي گويد آيا دو عمليات به صورت ذره وا انجام مي شود يا نه.
در حال حاضر يكي از عملياتي كه در چند پردارنده هاي قديمي استفاده مي شود تست كردن و نشاندن است (test-and-set) كه يك مقداررا تست مي كند و اگر آن مقدار توسط آن تست تصويب شد آن را قرار مي دهد. براي مثال ما مي توانيم عملياتي را تعريف كنيم كه براي ۰ تست شده و در آن ارزش ۱ قرار گرفته.نوع ديگر از همزماني اتميك او fetch a increment است كه ارزش محل حافظه و افزايش ذره اي را برميگرداند وجود ۰ نشان مي دهد كه متغير همزماني مطالبه نشده و ما مي توانيم از fe tch a increment فقط در مبادله استفاده كنيم كاربردهاي ديگري از عمليات وجود دارد مشابه fetch a increment كه مختصراً به آنها خواهيم پرداخت. دستورات بي وقفه در اجراي عمليات حافظه اتميك،زمانيكه به هر دو حافظه خواندني و نوشتني نياز است يكسري رقابتها را مطرح مي كند. پيچيدگي كه در كاربرد آن است مربوط به زمانيست كه سخت افزار هيچ عمليات ديگري را در بين خواندن و نوشتن نمي تواند انجام دهد و منجر به بن بست مي شود.
يك تبديلي در يك جفت دستور است زمانيكه دومين دستور ارزشي را برمي گرداند و مي توان نتيجه گرفت كه اگر اتميك بود آيا آن جفت دستور اجرا مي شد و زماني آن جفت دستور مؤثر هستند كه هيچ پردازنده ديگري ارزش را در بين آن جفت دستور تغيير ندهد.
اين جفت دستور يك load ويژه است كه lood locked , load linked را شامل مي شود و دستور ديگر يك store ويژه است كه store conditianad ناميده مي شود اين دستورات بترتيب استفاده مي شوند:اگر محتويات مكان حافظه با load liaked مشخص شود آن قبل از دستور store condionad كه با آدرس يكسان رخ داده تغيير پيدا مي كند. پس دستور store شرطي از بين مي رود و اگر پردازنده يك سوئيچ ميان آن دو دستور انجام دهد باز هم store شرطي از بين مي رود.

دستور store شرطي اگر انجام شود ۲ را باز مي گرداند در غير اينصورت ۰ را برمي گرداند و تنها زماني عمليات موفقيت آميز است كه load linked مقدار اوليه نرا برگرداند و store شرطي هم مقدار ۱ را بازگرداند. رشته زير يك مبادله اتميك را روي مكان حافظه مشخص شده بوسيله R1 انجام مي دهد:

try:Mov R3,R4 :Mov of value exchange

LL R2,0(R1) :loud linked

Sc R3,0(R1) :store condi tionad

BEQZ R3,try :branch store fails

Mov R4,R2 :put lood value in R4

در پايان اين رشته،محتويات R4 و مكان حافظه با R1 مشخص مي شود(با ناديده گرفتن هر اثري از branch هاي به تأخير افتاده).
در هر زمان يك پردازنده مداخله مي كند و مقدار حافظه را ميان دستورات LL و SC تغيير مي دهد sc مقدار ۰ را در R3 مي گذارد و باعث ترتيب كه براي try مي شود. يك مزيت مكانيزمهاي LL/SC اين است كه مي توانند براي ساخت همزمانيهاي اوليه ديگر استفاده شوند. به عنوان مثال در زير به يك fetch 8increment اتميك اشاره مي شود:

try: LL R2.0(R1) :load linked

DADUI R3,R2,#1 :i increment

SC R3,0(R1) :store conditionad

BEQZ R3,try :branch store fails

اين دستورات بوسيله نگهداري خط آدرس مشخص شده اجرا مي شوند براي دستور LL اگر يك وقفه رخ دهد يا اگر بلوك كش تطابق پيدا كند آدرس در link registet از بين مي رود (مثلاً به وسيله يك SC ديگر) دستور SC براحتي آدرس خود در لينك رجيستر را بررسي مي كند اگر بود در اينصورت SC موفقيت آميز بوده در غير اينصورت از بين رفته.
زمانيكه SC از بين مي رود بعد از دستور store ناتمام به آدرس LL بايد در انتخاب دستورات جايگزين شده بين اين دو دستور دقت كنيم كه دستورات رجيستر در آن مجاز به استفاده بوده و بدون خطرند،غير از اين ممكن است بن بست بوجود آيد(زمانيكه آن پردازنده نمي تواند دستور scsc را كامل كند) همچنين تعداد دستورات ميان S C.LL بايد كم باشد چون امتحان رخداد غيرمنتظره يا تكرار خرابي SC وجود دارد.

اجراي قفلهاي به هم پيوسته

قبلاً ما عمليات اتميك داشتيم و مي توانستيم از مكانيزمهاي به هم پيوستگي در يك چند پردازنده استفاده كنيم در اجراي قفلهاي چرخشي (spin lock) در يك حلقه آنقدر مي چرخد تا اينكه به نتيجه برسد.spin lock زماني استفاده مي شود كه برنامه نويس ها مي خواهند قفل را براي مدت كوتاهي نگهداري كنند و در مرحله اي با تأخير اندك از آن استفاده كنند پس بايد در دسترس باشد چون spin lock پردازنده را حبس مي كند و در يك l oop منظر مي ماند تا قفل آزاد شود البته در بعضي مواقع نامناسب هم هستند.

ساده ترين كاربرد آن مربوط به زماني است كه كش به هم پيوسته نباشد و قفلهاي متغير را در حافظه نگهداري مي كند. يك پردازنده دائماً تلاش مي كند تا قفل را در يك عمليات اتميك پيدا كند و تست كند كه آيا آن مبادله قفل را آزاد مي كند يا نه براي آزاد سازي قفل، پردازنده مقدار ۰ را ذخيره مي كند در اينجا يك رشته كه براي قفل چرخشي داريم كه آدرسش در R1 در يك مبادله اتميك استفاده مي شود:

DADDUI R2,R0.#1

lockit: EXCH R2,0(R1) ,atomic exchange

?BNEZ R2,lockit ,already locket

اگر پردازنده ما كش بهم پيوسته را پشتيباني كند ما مي توانيم كش قفلهاي مورد استفاده آن مكانيزم بهم پيوسته براي مقدار قفل مربوطه نگهداري كنيم. كش بودن قفلها دو مزيت دارد:اول اينكه اجازه مي‌دهد كه مرحله spining (تلاش براي تست و بدست آوردن قفل در حلقه) كه در كش كپي مي شود سريعتر باشد از دستيابي حافظه كلي روي هر يك از دريافتهاي قفل كه در هر جستجو نياز است. دومين مزيت آن مربوط به لوكاليتي دستيابي قفل است يعني اينكه پردازنده اي كه قفل را استفاده كرده درآيند نزديك دوباره آن را استفاده مي كند. در اين مورد ممكن است مقدار قفل دركش پردازنده باشد و در نتيجه زمان پيدا كردن قفل كاهش زيادي پيدا كند.

با كسب اولين مزيت نياز داريم كه يك تغيير روي پروسه چرخشي ساده بدهيم هر جستجو براي مبادله كردن در حلقه مستقيماً نياز به عمل نوشتن دارد اگر پردازنده هاي چندگانه در حال جستجو براي بدست آوردن قفل هستند هر يك نوشتني را توليد مي كنند و بيشتر اين نوشتن ها منجر به writemiss مي‌شود.

بدين ترتيب ما بايد پروسه قفل چرخشي را اصلاح كنيم بطوريكه با انجام عمل خواندن روي كپي موضعي قفل بچرخد تا اينكه آن قفل دستيابي شود جستجو براي بدست آوردن قفل با انجام يك عمليات جانشيني انجام مي شود يك پردازنده ابتدا متغير قفل را تست وضعيت مي كند و آن پردازنده مي خواند و تست مي كند تا اينكه با توجه به مقدار بدست آمده مشخص شود كه قفل باز شده.

آن پردازنده سپس بر خلاف همه پروسه هاي مشابه (در نوشتن چرخشي) عمل مي كند تا ببيند كه چه كسي متغير را ابتدا قفل مي كند همه فرايندها از يك دستور مبادله اي استفاده مي كنند كه مقدار قديمي را مي خواند و ۱ را درون متغير قفل ذخيره مي كند تنها برنده مقدار ۰ را مي بيند و بقيه ۱ را. پردازنده برنده كد را بعد از قفل اجرا مي كند و وقتي كه تمام شد مقدار ۰ را درون متغير قفل براي آزاد كردن قفل ذخيره مي كند. در اينجا كدهايي كه spin lock را اجرا مي كند مي بينيم(۰ براي باز شدن و ۱ را براي قفل كردن)

```
lockit : LD R2,0(R2) , load of lock
BNEZ R2,lockit , not available-spin
DADDUI R2,R0,#1 , load locked valure
EXCH R2,0(R1) , swap
BNEZ R2,lockit , nranch if lock washt 0
```

حال بررسي مي كنيم كه برنامه spin lock چگونه از مكانيزم به هم پيوسته كش استفاده مي كند شكل ۴,۲۳ نشان مي دهد پردازنده و گذرگاه يا عمليات فهرستي براي فرايندهاي چندگانه متغيري كه از مبادله اتميك استفاده مي كند را قفل مي كنند يكي از پردازنده ها ۰ را در قفل ذخيره مي كند و بقيه كش ها نامناسب بوده و مقدار جديد را براي به هنگام سازي كردن كپي آن در قفل واكشي مي كنند در اين كش كپي باز شدن قفل را با ارزش ۰ نشان مي دهد و بعد مبادله را انجام مي دهد.
اين مثال مزيت ديگر LL/SC را نشان مي دهد :

عمل خواندن و نوشتن به وضوح از يكديگر جدا شده اند.LL موجب هر ترافيك گذرگاهي نيست اين واقعيت نشان مي دهد كه آن رشته كد ساده همان مشخصات بهينه شده نسخه استفاده از مبادله را دارد (R1 آدرس قفل را نگه مي دارد،LL جايگزين LD مي شود و SC جايگزين EXCH

.

```
Lockit : LL R2,0(R1) , load linked
BNEZ R2,lockit ,not available-spin
```

DADDUI R2,R0,#1 ,locked value

SC R2,0(R1) ,store

BEQZ R2,lockit ,branch if store fails

اولين شرط فرم حلقه چرخشي را مشخص مي كند و دومين شرط رقابتي كه دو پردازنده سر دسترسي همزمان به قفل دارند را حل مي كند.
هر چند برنامه قفل چرخشي ساده است اما پيچايش بالاي آن در بكار بردن تعداد زياد پردازنده ها مشكل است چون ترافيك توليد شده مربوط به زماني است كه قفل آزاد شده است.

مدلهايي از حافظه هاي پايدار
وابستگي كش باعث مي شود كه پردازنده هاي چندگانه به نمايش پايداري حافظه بنگرند حال سؤال اين است كه پايداري چگونه است و چه موقع يك پردازنده بايد مقدارش توسط پردازنده ديگر update از آنجايي كه پردازنده ها از طريق متغيرهاي به اشتراك گذاشته ارتباط برقرار مي كنند پس اين سؤال را مي توان اينگونه مطرح كرد: در چه دستوراتي بايد به داده هاي نوشته شده توسط پردازنده هاي ديگر توجه كرد؟ در جايي كه تنها راه به واسطه خواندن است.
سؤال را مي توان بدينصورت گفت:چه چيزي را بايد بين خواندن و نوشتن در پردازنده هاي متفاوت اجرا كرد؟در ميان اين سوالات پايداري حافظه كه به نظر مي رسيد بايد ساده باشد فوق العاده پيچيده مي شود.

بعنوان مثال با يك مثال ساده مي توان آن را بيان كرد در اينجا دو كد سگمنت از پردازنده P1 , P2 است كه پهلو به پهلوي يكديگر نشان داده شده اند.

P1 : A=0 P2:B=0

....

A=1 B=2

.....(L2:if (B=0)..... L2:if (A=0

با فرض اينكه پروسه هايي در پردازنده هاي مختلف در حال اجرا هستند و محل هاي B , A , از دو پردازنده ابتدا در كش مقدار ۰ را دارند. دستورات (برچسب دارL1, L2) شرطها را ارزيابي مي كنند و اگر درست بود در اينصورت B, , A مقدار ۱ را كه به آن اختصاص داده ايم نشان مي دهند. اما فرض كنيد كه آن نوشتن هاي اضافي به تأخير افتاده باشد و آن پردازنده ها در طي آن تأخير به پردازش ادامه دهند. و ممكن است كه P 2 ,P1 مقداري براي B, A نداشته باشد(بترتيب) و قبل از اينكه شروع به خواندن مقدار ها كند سؤالي كه پيش مي آيد اين است كه چه شرايطي لازم است تا اين وضعيت ادامه داشته باشد؟آسانترين مدل براي پايداري حافظه پايداري ترتيبي sequential consistency ناميده مي شود.
در پردازش ترتيبي بايد نتيجه هر اجرايي مشابه باشد همچنين دستيابي هاي حافظه كه توسط هر پردازنده انجام مي شود به طور صحيح نگهداري مي شود و اين دستيابي هاي پردازنده هاي مختلف به دلخواه جايگذاري مي شوند. پايداري ترتيبي امكان اينكه تعدادي از اجراها نامعلوم باشد را در مثال قبل رفع كرد چون آن انتصابها بايد قبل از ورود دستورات بعدي كامل شوند.

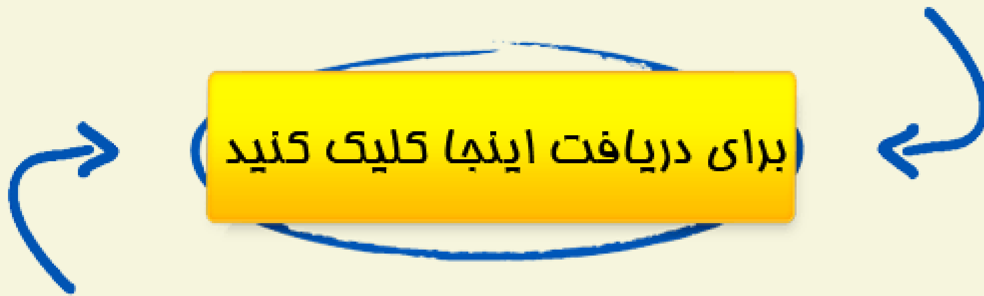
ساده ترين را در اجراي پايداري ترتيبي وجود پردازنده اي است كه با تأخير دستيابي حافظه را به اتمام برساند تا اينكه همه آن موارد از بين رفته با دستيابي كامل شود. و آن تأخير برابر با دستيابي حافظه بعدي است به ياد داريم كه پايداري حافظه عمليات را با متغيرهاي مختلف درگير مي كرد:

آن دو دستيابي كه بايد مرتب باشند در واقع مكانهاي متفاوت حافظه هستند در اين مثال ما بايد تأخير A يا B را با تأخير انجام دهيم (B=0 يا A=0) تا اينكه نوشته قبلي كامل شود(A=1 يا B=1) در نهايت اينكه در پايداري

ترتيبي ما عمل خواندن و نوشتن را با هم نمي توانيم انجام دهيم هر چند پايداري ترتيبي يك شكل ساده از برنامه نويسي را ارائه كرد اما كارائي را پايين آورد. حال به يك مثال ديگر مي پردازيم.

مثال:تصور كنيد كه ما پردازنده اي داريم كه write miss آن so سيكل براي برقراري مالكيت مي گيرد،۱۰ سيكل براي توليد هر ابطالي بعد از برقراري پايداري نياز است ۸۰ سيكل براي كامل شدن يك ابطال و تصديق اينكه توليد شده است فرض مي كنيم كه يك بلوك كش بين چهار پردازنده تقسيم شده است حال چقدر write miss بايد توقف داشته باشد تا پردازنده نوشته شود؟(البته اگر پردازنده پايدار ترتيبي دارد.)و نيز فرض مي كنيم كه قبل از اينكه كنترل كننده هاي وابسته كامل شود بايد باطل سازي صريحاً تصديق شود چنانچه ما بعد از بدست آوردن مالكيت براي write miss بتوانيم اجرا را دامه دهيم بدون اينكه براي باطل سازي صبر كنيم پس نوشتن چقدر طول مي كشد؟

پاسخ: وقتي كه ما براي باطل كردن صبر مي كنيم هر نوشتني به اندازه زمان مالكيت وقت مي گيرد و وقتي كه باطل سازي ها روي هم مي افتند ما فقط در مورد آخرين آن نگران مي شويم كه اين با۱۰+۱۰+۱۰+۱۰=۴۰ سيكل بعد از اينكه مالكيت برقرار شده آغاز مي شود سپس زمان كل براي هر نوشتن ۵۰+۴۰+۸۰=۱۷۰ سيكل است در اين مقايسه زمان مالكيت se سيكل است.



مقالات مرتبط

- دانلود مقاله اوقات فراغت
- دانلود مقاله ماشینهای جوشکاری جریان مستقیم
- دانلود مقاله ماشینهای الکتریکی

از این سایت ها نیز دیدن نمایید

- ترنس لاین ، مرجع مقالات تخصصی فارسی ایران
- گت پیپر ، منبع مقالات انگلیسی و فارسی
- دانش رسان ، بیش از 1.5 میلیون مقاله فارسی