



تأثیرات تجزیه و تحلیل پیکر بندی حافظه نهان در هنگام پیش بینی و تشخیص انشعابات غیر مستقیم

فاطمه حیدری، سمانه فتاحی، حسین بیگی

دانشگاه آزاد اسلامی واحد دولت آباد

دانشگاه آزاد اسلامی واحد دولت آباد

دانشگاه آزاد اسلامی واحد دولت آباد

Heidari_6f6@yahoo.com, Hossein_beigi@yahoo.com

چکیده

این مقاله تأثیرات استفاده از حافظه نهان را روی پیش بینی انشعابات غیر مستقیم در پردازنده های ILP مورد بحث قرار می دهد و سهم اصلی آن شناخت این واقعیت است که محتوای حافظه نهان حاوی اطلاعاتی درباره جریان کنترلی برنامه اخیر است که می تواند دقت پیش بینی کننده ها را که خودشان قادر به استفاده صریح از چنین اطلاعاتی نمی باشند، افزایش دهند. ما نشان می دهیم که آدرس مسیر انشعاب غیر مستقیم در حافظه نهان دقت پیش بینی انشعاب غیر مستقیم را افزایش می دهد. پس بصورت تدریجی فشردگی سازی، اضافه کردن شمارنده بیتی در هر خط حافظه نهان شرکت پذیری حافظه نهان، سبک آن و... را برای تأثیر هر پیکر بندی روی پیش بینی انشعاب غیر مستقیم مورد بررسی قرار داده ایم. این مقاله تکنیک جدیدی را برای استفاده کافی از trace cache های کوچک ارائه می دهد. یک trace cache می تواند کارایی پردازنده ها را بطور قابل توجهی افزایش دهد. trace cache ها بعنوان راه حلی برای مساله مکانیسم واکنشی در پردازنده های اخیر بطور موثری مورد استفاده قرار گرفته اند.

کلمات کلیدی

integer، trace cache، انشعابات غیر مستقیم

۱-مقدمه

موتور اجرای گسترده به واحد fetch (واکنشی) دستور العمل بزرگتری نیاز دارد، که باید پهنای واحد کاربردی آن ها به منظور استفاده بهینه از همه آنها با هم match (هماهنگ) باشد.

پردازشگرهای (ILP)، دستورالعمل موازی سطح، خواه superscalar و یا VLIW بر واحد های کاربردی گسترده ای برای استخراج ILP از application ها دارد.

های تابعی کمتری می باشند، این است که انشعابات قابل پیش بینی باشند. اگرچه، فرکانس استفاده از انشعابات غیر مستقیم بیشتر از کاربردهای آینده است، اگر چه انشعابات شرطی در بیشتر application ها، ادامه می یابند. برای مثال برنامه های شیء گرا object oriented، از انشعابات غیر مستقیم بیشتری برای پیاده سازی فراخوان های تابع مجازی استفاده می نمایند. کتابخانه های نیک شده بصورت دینامیکی (dll) با استفاده از انشعابات غیر مستقیم فراخوانی می شوند. نهایتاً افزایش تعداد زبانهای برنامه نویسی با استفاده از ماشین های مجازی (VMS) نظیر java به تکنیکی نیاز دارند که تعداد زیادی انشعاب غیر مستقیم را دارا باشد. بنابراین نرخ دقت در پردازنده های ILP افزایش می یابد.

ما مشاهده می کنیم که در یک پردازنده ILP با Trace cache، امکان افزایش دقت پیش بینی انشعاب غیر مستقیم برنامه ها با استفاده از Trace cache به جای استفاده از بافر مسیر انشعاب وجود دارد. بعلاوه، استفاده از این طرح مستلزم ایجاد ساختار بسیار ساده سخت افزاری بدون استفاده از جداول اضافی می باشیم.

در این مقاله نشان می دهیم چگونه Trace cache می تواند برخی از محتوای اطلاعاتی مورد استفاده بوسیله پیش بینی کننده های غیر مستقیم دوطبقی را تقسیم کند. اگر چه افزایش دقت، کمتر از انشعاب غیر مستقیم دو سطحی می باشد و هزینه نیز کمتر است. اگر کسی قصد ساخت یک Trace cache را داشته باشد، باید قادر باشد پیش بینی غیر مستقیم را با یک BTB نرخ کم با هزینه غیر اضافی انجام دهد. ما در این مقاله برای مقایسه پیش بینی کننده های انشعاب غیر مستقیم دو سطحی یا nent trace تلاشی نکرده ایم. ما منحصراً مزایای دو سطحی را با استفاده از Trace cache نشان داده ایم. مبنی بر مشاهدات اولیه را هکارهایی را برای افزایش دقت ارائه می دهیم (۱) پیشنهاد می کنیم آدرس مسیر انشعاب غیر مستقیم را در Trace cache بهنگام سازیم. ما استفاده از سایت بهنگام سازی را پیشنهاد می نماییم (۲) نشان می دهیم بهنگام سازی با شمارنده ۲ بیتی در انتهای هر خط انجام گیرد.

2-2-بخش دوم

طبق شکل ۱، پردازنده های superscalar با کارایی بالا به مکانیسم واکشی دستورالعمل و مکانیسم اجرای دستورالعمل تقسیم می شود. موتور اجرای دستورالعمل مصرف کننده ای است که دستورالعمل ها را از بافر حذف

در واقع واحد واکشی دستورالعمل باید تعداد زیادی از دستورالعمل ها را از cache میان بلاک های اصلی در هر سیکل، واکشی کند. عموماً این بلاک های اصلی جایگزین موقعیت های غیر پیوسته در cache می شوند. بیش از یک دسترسی به cache باید به منظور واکشی همه بلاک های اصلی مورد نیاز، صورت گیرد. مکان های واکشی غیر پیوسته در تعداد دستورالعمل ها محدود است. از این رو cache را همکار مناسبی برای واکشی چندین بلاک اصلی در هر سیکل محسوب می گردد.

Trace cache (اثر کش) تکنیک ریز معماری برای افزایش پهنای باند واکشی دستورالعمل یک پردازنده superscalar می باشد. پردازنده، جریان دستورالعمل اجرا شده را جمع آوری کرده و آنها را در خطوط Trace cache با آدرس های مسیر انشعاب ذخیره می نماید. در این روش، دستورالعمل های واکشی شده، از مسیر اجرای غیر پیوسته در موقعیت های پیوسته در Trace cache جایگزین می گردد. هنگامی که یک خط Trace cache به مسیر اجرای بعدی، ارجاع داده می شود، خط کلی را می توان در طی یک دسترسی به cache واکشی نمود. [1]. مکانیسم های واکشی و اجرا بوسیله بافرها برای مثال صف ها، مکانیسمهای رزرو و... مجزا شده اند. مکانیسم واکشی دستورالعمل مانند یک تولید کننده ای که عمل واکشی دیک و غیره را انجام می دهد، عمل می کند. [2] کارایی یک پردازنده تک thread به شدت وابسته به میانگین تعداد دستورالعمل های مفیدی است که می توان آنها را در سیکل واکشی نمود. trace cache ها دربرآورده سازی این هدف با ذخیره دستورالعمل ها بصورت پویا بسیار مؤثر هستند [3]. trace cache ها بعنوان راه حل موثری برای مشخص کردن محدودیت های مکانیسم واکشی پیشنهاد شده اند. این cache ها بلاک های اصلی دستورالعمل را به روشی که مجدداً در طی اجرای برنامه قابل بازیابی باشند، ذخیره می کند. هر خط trace یک دنباله از بلاکها را با اطلاعات نزدیک به بلاکها ذخیره می نماید. برای پیش بینی انشعاب بعدی در طی هر واکشی چندین پیش بینی کننده انشعاب مورد استفاده قرار گرفته است. [4] در این مقاله، ما به ترتیب بخش های زیر را دنبال می کنیم:

2-1-بخش اول

عموماً، هدف اصلی C و برنامه های Fortran که دارای انشعابات غیر مستقیم کمتر، انشعابات شرطی و بازگشت

تنظیم دستورالعمل غیر پیوسته - بدلیل انشعابات و پرش ها jumpها ، و دستورالعمل ها باید در طی هر سیکل داده شده واکنشی گردند . از این رو ، باید مسیر های مناسب و منطقی برای واکنشی و تنظیم بلاک های اصلی غیر پیوسته و عبور آنها به گذرگاه وجود داشته باشد . فقط کافی نیست که دستورالعمل ها در cache حضور داشته باشند ، بلکه باید آنها بصورت موازی قابل دسترسی باشند .

دوام واحد واکنشی ، دوام گذرگاه تاثیر مهمی روی کارایی پردازنده دارد . این تاثیر ناشی از هزینه پر کردن مجدد گذرگاه بعد از کنترل غیر صحیح می باشد . درمورد واجد واکنشی ، هزینه شروع واکنشی مجدد بعد از کم کردن انشعاب ، Jump یا دستورالعمل cache بسیار مهم است . نیاز به بازدهی انشعاب بیشتر و تنظیم دستورالعمل غیر پیوسته دوام واحد واکنشی افزایش خواهد یافت .

واحد های واکنش کنونی به یک پیش بینی انشعاب درهرسیکل محدود شده می تواند ۱ بلاک اصلی را در هر سیکل واکنشی کند .

داده های موجود درجدول ۱ ، میانگین ساینز بلاک های اصلی را که در حدود ۴ یا ۵ دستورالعمل برای کدهای صحیح می باشد ، نشان می دهد . واکنشی یک بلاک اصلی درهر سیکل برای پیاده سازی بیش از ۴ دستورالعمل درهر سیکل کافی می باشد . اگر ما چندین پیش بینی انشعاب را معرفی می نمایم ، واحد واکنشی در نهایت می تواند چندین بلاک اصلی پیوسته را درهر سیکل واکنشی نماید . داده هایی که برای چندین دستورالعمل میان انشعابات گرفته شده مورد استفاده قرارمی گیرند تا حدی به دلی فرکانس انشعابات محدود شده است . بنابراین ، اگر یک انشعاب گرفته شود ، واکنشی دستورات زیرمسیر قبلی در همان سیکلی که انشعاب واکنشی شده است ، مورد نیاز می باشد .

جدول (۱) : انشعاب و بلاک اصلی

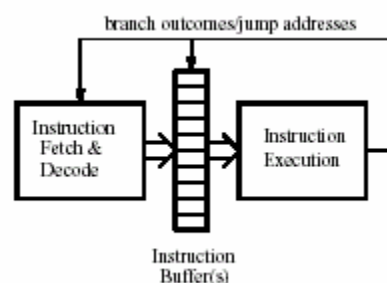
Benchmark	taken %	avg basic block size	# instr between taken branches
eqntott	86.2%	4.20	4.87
espresso	63.8%	4.24	6.65
xlisp	64.7%	4.34	6.70
gcc	67.6%	4.65	6.88
sc	70.2%	4.71	6.71
compress	60.9%	5.39	8.85

2-3-بخش سوم

استفاده از trace cache ، اگر دارای ساینز محدود شده ای نباشد ، بسیار مؤثر است . اگر چه trace cache های

کرده و سپس آنها را اجرا می کند . وابستگی های کنترلی (انشعاب ها و jumpها) یک مکانیسم باز خورد را میان تولید کننده و مصرف کننده بوجود می آورد .

پردازنده هایی که از این تکنیک ها استفاده می نمایند ، سعی در اجرای دستورالعمل بصورت موازی با هم دارند . بافر های بزرگ برای حفظ یک قالب دستورالعملی برای اجرا بصورت موازی مورد استفاده قرار می گیرند . برای فعال نمودن اجرای همروند دستورات ، موتور اجرایی از واحد های کاربردی زیادی که بصورت موازی با هم قرار گرفته اند عمل می نماید . موتور واکنشی انشعاب ها را برای اجرای پیوسته دستورات به درون قالب window ارسال می کند . هدف از این طرح افزایش مقیاس این تکنیک هاست :



شکل ۱: تجزیه موتورهای عمل گر واکنشی

Dispatch وسیعتر ، قابلهای بزرگتر ، ثابت های فیزیکی بیشتر، واحد های کاربردی بزرگتر و برای حصول این اهداف ، برقراری تعادل میان همه قسمتهای پردازنده بسیار مهم است .

در این مقاله ، ما پهنای باند واکنشی دستورالعمل را برای افزایش کارایی مورد بررسی قرار داده ایم . کارایی واکنشی دستورالعمل وابسته به تعدادی از فاکتورها می باشد . نرخ اصابت و دستورالعمل cache و دقت درش بینی انشعاب مسائل بسیار مهمی است که در چنین واکنشی دستورالعمل مورد استفاده قرار می گیرد . در این مقاله ، فاکتورهای دیگری را که در هر سیکل واکنشی مؤثر است بررسی نموده ایم :

بازدهی انشعاب : اگر فقط در هر سیکل یک انشعاب شرطی اجرا شود ، می توان window را در نرخ فقط یک بلاک اصلی در هر سیکل افزایش داد . پیش بینی چندین انشعاب در هر سیکل باعث می شود بازدهی دستورالعمل کلی بالاتر گردد .

اگر `trace cache` حضور داشته باشد، آن از `trace cache` یک سیکل واحد بازیابی شده است. درحالیکه `trace cache` ها به حصول کارایی بهتر کمک میکنند اما دارای محدودیت هایی نیز می باشند. برخی از این محدودیتها عبارتند از:

- دیده شده است که با استفاده از `trace cache` های بزرگتر، مزایای مربوط به کارایی کاهش می یابد. درواقع مصرف توان افزایش می یابد.
- `trace` هایی که کاملاً شامل بلاک های پیوسته باشند احتمالاً درخطوط آدرس دهی `cache` موجود می باشند. حضور چنین `trace` هایی منجر به ایجاد افزونگی و هدر رفتن فضای حافظه خواهد شد.
- با توجه به اینکه ۸۰٪ زمان درزمینه ۲۰٪ از کدها صرف می شود، ممکن است LRU روش بهتری باشد.

راهکار ما برای حل این مسائل استفاده از نموداری است که برای طراحی جایگزینی `trace` ها در `trace cache` مورد استفاده قرار می گیرد. هربرنامه کاربردی به پارتیشن های کوچکتر تقسیم می شود و هریک ازاین زیرمجموعه ها نمودار می شوند لوکالیتی درجریان نمونه افزایش می یابد.

کاربرد اصلی این پروسه تعریف و تشخیص `trace` هایی است که غالباً مورد استفاده قرار گرفته است. این پروسه درواقع برای تقسیم `trace` ها به دو سطح بصورت زیربسیار مهم هستند هرزمانی که یک `trace` به دورن `trace cache` آورده می شود جزئیات آن با `trace` های موجود درجدول چک می گردد.

اگر اطلاعاتی یافت نشود، `trace` به دورن `cache` وارد شده و شمارنده آن را تغییر می یابد. بنابراین هروقت `trace` مورد اصابت قرار می گیرد، شمارنده انیز افزایش می یابد. هنگامی که درصد استفاده آن به آستانه `t1` می رسد، جزئیات آن به یک مدخل در `cache` نوشته می شود. اگر بعد ازاین `trace` از `cache` بازیابی شود، مدخل جدول به آخرین مقدار بهنگام سازی می گردد. اگر بعد ازاین زمان، همان `trace` به `cache` وارد شود، این جزئیات به شمارنده انتقال داده می شود.

هنگامی که مقدرا شمارنده به دومین آستانه `t2` برسد، شمارنده بهنگام سازی می گردد. توجه کنید که شمارنده استفاده از آنها دارای مقادیر بزرگتر از `t2` می باشد، قادر به حذف شدن نمی باشند. اگر جدول با چنین مداخلی پرشود، ماشرو به درخواست برای هر مدخل می کنیم. هنگامی که شمارنده به ۱۵ برسد، پروازنده متوقف شده و

کوچک به علت زمان دسترسی و کارایی حافظه بسیار کاربردی تر است: بلاک های اصلی را می توان در `trace cache` های مختلفی مشاهده نمود، بعضی از `trace` ها حاوی ماکزیمم دستورالعمل ها نمی باشد و بلاک مورد در خواست ممکن است درون `trace` باقی مانده و غیر قابل دسترس گردد. بنابراین کلید برای استفاده مؤثر از `trace cache` های بسیار کوچک، ارائه روشی برای تغییر ساخت آنهاست. این مقاله برروی دومین قسمت منطقی مورد نیاز برای ساخت `trace` ها متمرکز شده است. فیلترینگ هم اکنون روش پیشنهادی برای افزایش تاثیر سایز `trace cache` میباشد. دراین روش پیشنهاد می شود، همه `trace` هایی که حاوی انشعابات هستند ذخیره شوند. پس باید آنها را مبنی بر درصد استفاده نشان فیلتر کرد. `trace cache` به دو بلاک اصلی تقسیم شده است: (FTC)، (MTC). همه `trace` ها روی FTC نوشته می شود. اما فقط `trace` های موجود در MTC مفید می باشند. موفقیت این روش فیلترینگ مبنی برمشاهده آنهاست، بیشتر `trace` های ساخته شده و قرار گرفته در `trace cache` قبل از `eviction` مورد استفاده قرار گرفته و بیشتر دستورالعمل ها از مجموعه `trace` های کوچک اجرا می شود. پیشنهاد شده است که به منظور فیلتر کردن `trace` هایی که غالباً مورد استفاده قرار گرفته اند، نمونه برداری شده و لوکالیتی آنها نشان داده شود. این مقاله نوع جدیدی از تکنیکهای فیلترینگ را که مبنی برنمونه برداری آماری `trace` ها می باشد معرفی می نماید. این نوع فیلتر ها می خواهند کیفیت `trace` ها را دریک `trace cache` کوچک افزایش دهد. این مقاله کارایی و توان فیلتر نمونه برداری اصلی (sf) وورژن ارتقاء یافته آن را به همراه مقایسه آن با `ftc` - `mtc` تحلیل و بررسی می کند.

2-4-بخش چهارم

پردازنده های `superscalar` قادر است تعداد زیادی از دستورالعمل ها را در قالب یک سیکل واحد اجرا نماید. اگر چه مکانیسمهای واکنشی دستورالعمل باید دستورالعمل ها را با مکانیزم موازی اجرا نماید. دقت پیش بینی انشعاب، نرخ اصابت در `cache` و قواسیهای اجرای دینامیکی فاکتور های تاثیر گذار در کارایی مکانیسم واکنشی می باشند.

آدرس بلاک شروع و بیت های آن برای تعیین حضور `trace` مورد نیاز در `trace cache` مورد استفاده قرار گرفته است.

4-2- طرح جدول

این طرح شناسایی مهمترین trace ها در یک از پارتیشن های برنامه را انجام می دهد. هنگامی که نمونه درجراهای متوالی مورد استفاده قرار گیرد، این نمونه باید قادر باشد به ما کمک کند تا trace های مهم و غیر مهم را از هم تشخیص دهد. بازیابی trace های مورد استفاده کمک می کند تا سر باریسیستم در هنگام ساخت مجدد trace کاهش یابد، همچنین درصد هرز رفتن فضا در cache نیز کاهش می یابد. این نتیجه از این حقیقت مشتق شده است که شانس های زیادی موجودند که trace هایی را مورد استفاده قرار میدهند که ممکن است جایگزین trace های مورد استفاده متداول موجود در cache گردند. محققان معتقدند که راهکار پیشنهاد شده تنها hit های trace را افزایش نمی دهد، آنها می توانند منطق بافرپر را ذخیره کنند.

4-3- ساخت جدول

جدول باید جزئیات را درباره trace های متداول و به روش فشرده دارا باشد. Trace کامل شده

هنگامی که یک trace برای اولین بار به درون trace cache آورده می شود، شمارنده درصد استفاده آن با ۱ مقدار دهی اولیه می گردد. با هراسابت این شمارنده افزایش می یابد و وقتی مقدار آن به بالا تر از مقدار استانه t1 افزایش یافت، یک مدخل برای این trace ساخته شده و همه جزئیات نظیر آدرس بلاک شروع، بیت های نمونه انشعاب و شمارنده درصد استفاده به درون آن نوشته می شود. بعد از اینجا اصابتها شمارنده های درصد استفاده را در trace cache و جدول بهنگام سازی می کند. حتی اگر این trace از cache حذف شود، مدخل جدول بدون آسیب باقی می ماند. در زمان بعدی همان trace آورده می شود شمارنده مربوط به درصد استفاده آن با مقداری جدول، مقدار دهی اولیه می گردد. و با هراسابت مقدار آن افزایش می یابد. هنگامی که این درصد استفاده به مقدار استانه t2 برسد، تصمیم میگیریم که matvre trace شود و بیت mask مدخل آن در جدول با تنظیم شود. توجه داشته باشید که همه مداخل باید update گردد. استراتژی وقفه که ما مورد بحث قرار دادیم مارا درساختن جدول برای هر پارتیشن برنامه یاری می نماید.

اطلاعات به دورن فایل حافظه نوشته می شود. در این روش کل برنامه مورد استفاده قرار گرفته است. منطق جایگزینی مبنی بر اهمیت trace ها به نسبت یکدیگر است. trace هایی که شامل انشعابات برگرفته باشد. بعنوان بهترین کاندیدها برای جایگزینی در نظر گرفته می شوند.

سر باریسیستم روی سخت افزار اعمال می شود. در این مقاله مافقط طرح مربوط به معماری والگوریتم های مورد استفاده برای منطق جایگزینی را ارائه می دهیم.

3- طراحی وظایف

طرح پیشنهاد شده به انجام تغییرات مشخصی در زمینه طراحی موجود نیازمند است.

3-1- جدول نمونه

به منظور جدول بندی trace ها مابه یک جدول نمونه نیاز داریم که اطلاعات مربوط به trace در آنها حفظ و نگهداری گردد. این جدول حاوی مداخل بسیراز یادی می باشد که حاوی trace cache در زمان داده شده می باشد. هر مدخل باید فیلدهای زیر را دارا باشد:

4- آدرس های بلاک شروع

- بیت های پیشنهاد بینی مربوط به بلاک های موجود در trace ها
- فاکتور اولویت مربوط به درصد استفاده trace در هر یک از زیر مجموعه های برنامه. برای اجتناب از wrap around این فیلد باید دقت کافی صورت گیرد.
بیت mask برای اینکه بتوان یک مدخل را برای جایگزینی انتخاب نمود.

4-1- تغییرات trace cache

خط trace cache باید برای طرح پیشنهادی، دچار تغییرات و اصلاحاتی گردد. هر مدخل در trace cache باید شامل یک فیلد باشد تا درصد استفاده حفظ گردد. این فیلد مرتبط با فاکتور اهمیت اولین زمانی است که یک trace جدول بندی شده به درون trace cache آورده شده است.

یک بیت تصمیم میگیرد که trace مهم است یا اینکه دارای اهمیت نمی باشد درصد استفاده از این بیت به تفصیل بیان شده است.

4-4- جایگزینی مدخل جدول

طبیعی است که همه مدخل جدول نمونه پر شوند و روشن مناسبی برای جدول بندی دیگر *trace* هائی که ممکن است درصد اصابت به آنها از *t1* بیشتر باشد در نظر گرفته شود. ماشاخصی رابرای مداخل جدول ساخته و جایگزینی را انتخاب می نمایم.

4-5- استراتژی وقفه

همه مداخل مربوط به جدول نمونه تکمیل خواهد شد و هیچ مدخلی برای جایگزینی موجود نم ییادش از این رو همه درخواست ها برای مداخل جدید محدود می شود. اگر این امر اتفاق بیفتد ما را شمارنده ای استفاده می کنیم که عدد تکذیب ها را حفظ کند.

و هنگامی که مقدار آن به ۱۵ برسد، زمان آن به درون فایلی در حافظه نوشته می شود. وقفه تولید شده باید بوسیله پردازنده سرویس دهی شود. در هنگام هریک از نوشتن های نمونه همه مداخل *cache trace* باید به صفر برسد.

4-6- درصد استفاده نمونه

نمونه را می توان در زمان اجرا مورد استفاده قرارداد. البته اگر شرایط اجرای برنامه یکسان باشد.

هنگامی که محیط اجرا تغییر نکند، جدول مربوط به هر زیرمجموعه از برنامه به درون جدول نمونه بار گذاری می شود. هدف از این بحث این است که ما مجاز به تعریف و تشخیص *trace* های مهم و غیر مهم باشیم. قبل از اینکه یک خط بوسیله با فر ساخته شود. جدول نمونه برای تعیین یک *trace* مهم در مورد بررسی قرار می گیرد. اگر عمل انجام گیرد *trace* ساخته شده و سپس به درون

cache بارگذاری می شود، شمارنده درصد استفاده با فاکتور اهمیت تنظیم می شود و بیت اهمیت در *trace* تنظیم می شود. اگر عمل محدود صورت نگیرد یک مقدار تصادفی میان *t1,0* محاسبه می شود. این مقدار باید در شمارنده درصد استفاده *trace* ذخیره گردد، این عمل با تنظیم مجدد بیت اهمیت در صورتی که *trace* مدخل

cache را پیدا کند، انجام می گیرد. هر زمانی که *trace* در *cache* مورد اصابت قرار گیرد، فاکتور اهمیتش به ۱ کاهش می یابد. منطق جایگزینی بعداً بحث قرار میگیرد. هنگامی که همه خطوط *cache trace* با بیت اهمیتش تنظیم گردد. طرح زیرمورد استفاده قرار میگیرد. اگر خط ساخته شده با منطق بافر یک *trace* مهم باشد. *trace* با فاکتور نهایی اش فعال می گردد. مدخل مرتبط با این

trace در جدول برای منعکس کردن آخرین مقدار فاکتور اهمیت تغییر می یابد.

اگر یک *trace* غیر مهم با منطق بافر پر ساخته شود. یک مقدار تصادفی میان محاسبه می گردد. اگر مقدار تصادفی *trace* جدید بزرگتر از فاکتور اهمیت نهایی باشد، دومی با *trace* ساخته شده جدید جایگزین می گردد، در غیر این صورت *trace* جدید یک مدخل جستجو م یگردد و پس فاکتور اهمیتش با یک مقدار تصادفی محاسبه شده مقدار دهی اولیه می گردد. اگر در *cache* یک *trace* مهم یا بیش از یک *trace* مهم موجود باشد، یک ایندکس به *trace cache* جدید وارد می شود و *trace* ها به روش گردش نوبتی مورد جستجو قرار میگیرد.

در موارد بالا مقایسه بیت های پیشگویی میان *trace* هایی که دارای فاکتور اهمیت اولیه هستند، با شکست مواجه می شود.

trace هایی که شامل انشعابات برگرفته هستند، مقدم ترند. اگر بیت های پیش بینی *trace* ها مشخص باشد، یکی از آنها بطور تصادفی انتخاب می شود.

5- مزایا

در هنگام استفاده از منطق جایگزینی مزایای زیادی حاصل می شود. مزایای حاصله مربوط به افزایش کارایی *trace cache* ها می باشد. می توان دید که کارایی *trace cache* با افزایش سایز افزایش مییابد.

نتایج مورد استفاده در این بخش از طراحی *cache trace* برگرفته شده است. *ipc* نشان داده شده در اینجا میانگین برای یک مجموعه با برنامه نسبت م یباشد. افزایش سایز کش باعث می شود تعداد *trace* ها در هر نقطه از زمان در کش افزایش یابد. این افزایش یابد. این افزایش باعث زیاد شدن اصابتها و افزایش کارایی می گردد. اگر چه آشکار است که افزایش سایز اقتصادی نم یباشد. زیرا مصرف نیروی کش زیاد می گردد.

این طرح سعی دارد هنگامی که کش کاملاً پراز *trace* های متداول است، آنها را مورد استفاده قرار دهد. واکشی بعدی برای *trace* متداول باعث ساخت مجدد *trace* می گردد.

trace مورد استفاده باید خیلی زود از *cache* حذف گردد. مشاهده شده است که چنین استفاده افراطی از *cache trace* نباید در طرح ما مجاز باشد.

هنگامی که دو *trace* برای جایگزینی در کش با هم رقابت می کند. طرح ما اولویت رابه *trace* های می دهد که

[2]- james smith , steve Bennett , eric rotenberg

، علوم فرورفتگی کامپیوتری ، دانشگاه دیسکونسنین ، سال ۲۰۰۳

[3]- micaeel behar, Avi Mendelson , Avinomam

، دانشگاه مهندسی الکترونیک هیفا اسرائیل ، سال ۲۰۰۵ .

[4]- Saisuresh Krishnakmuaran

الکترونیک و کامپیوتر، کنیا هند ، سال ۲۰۰۴

زودتر ظاهر شده اند . دلیل این امر مبنی بر دو مشخصه زیراست .

۱ پیش بینی کننده های چندین انشعاب به دلیل اینکه مجبورند تعداد انشعابات را در طرح پیشگویی کنند ، دارای دقت کمتری می باشند .

6-نتیجه

در این مقاله ، تاثیرات استفاده از Trace cache را روی پیش بینی انشعاب غیر مستقیم در پردازنده های ILP مورد بررسی قرار داده ایم . اگر آدرس های مسیر در خطوط Trace cache برای پیش بینی انشعاب استفاده گردد ، دقت را می توان متفاوت از بافر آدرس ساده بررسی نمود . دلیل اصلی این است که Trace cache حاوی جریان کنترلی اخیر برنامه می باشد .

ما یک مکانیسم ساده بهنگام سازی آدرس های انشعاب غیر مستقیم در خطوط Trace cache ، ما دقت را می توانیم با صرف کمترین هزینه یا حتی بدون صرف هزینه افزایش دهیم . ما پیکربندی های متفاوتی را به همراه استراتژی های مختلف نظیر فشرده سازی ، اضافه کردن شمارنده ۲ بیتی ، مشارکت مجموعه cache ، ساینز cache ساینز خط cache و تاثیرات مثبت بر منفی هر پیکربندی بر استراتژی را مورد بررسی قرار داده ایم .

نتایج آزمایش ما متوسط دقت ها رمونیک را در میان چندین انشعاب با استفاده از یک مدل Trace cache به همراه پارامترهای Trace cache نشان داده شده است که می توان آن را با استفاده از بهنگام سازی نسبت به BTB ، ۳۵/۷۵٪ و نسبت به حالت غیر بهنگام سازی ۱۱/۷۷٪ افزایش داد .

اگر چه این مقاله تاثیر Trace cache را روی پیش بینی غیر مستقیم که ممکن است تاثیر مشابهی روی دیگر فرمها داشته باشد ، مورد بررسی قرار داده است . برای مثال ، مقدار بار ممکن است در صورت وجود مداخل مجزا برای هر یکی تحت تاثیر قرار گیرد . محاسبه این تاثیر برای طراح پردازنده که انتظار آدرس دهی با استفاده از Trace cache را دارد بسیار مهم است . [1]

مراجع

[1]- Davin Greeg ، ژورنال، برچسب دستورات برابر ،

دانشگاه ایرلند ، سال ۲۰۰۶